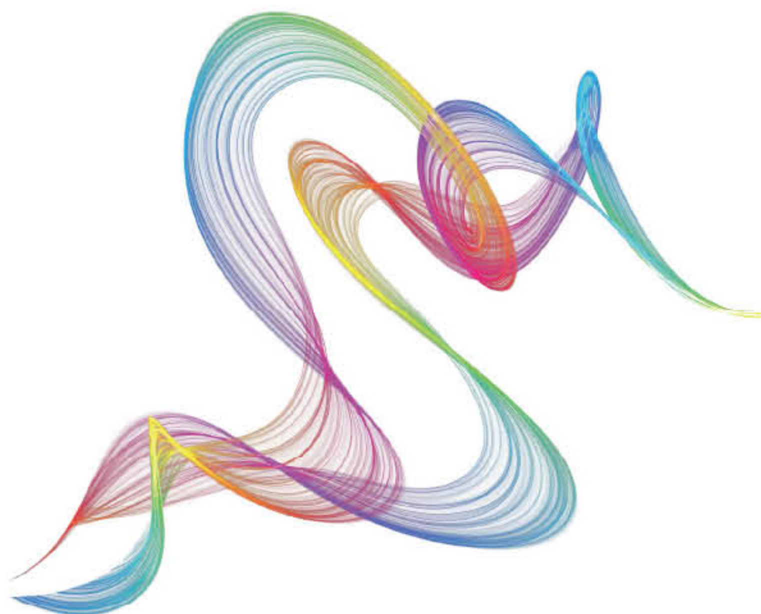
华章科技

PACKT PUBLISHING

内容全面而深入，既展示Mahout的强大功能，又全方位讲解利用Mahout进行大数据分类、聚类和预测分析的各种技术细节、方法和最佳实践

实战性强，包含丰富案例，涉及Mahout开发环境、序列文件使用方式、整合Mahout和外部资源、实现朴素贝叶斯分类器、股市预测、顶棚聚类、频谱预测、K–均值聚类等

大数据技术丛书



Apache Mahout Cookbook

Mahout实践指南

（美） Piero Giacomelli 著

靳小波◎译



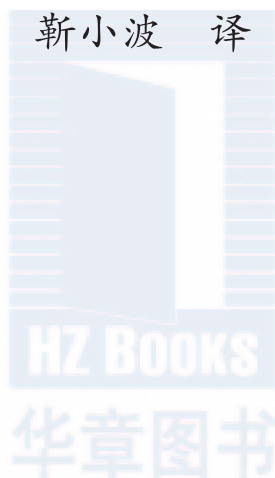
机械工业出版社
China Machine Press

大数据技术丛书

Mahout 实践指南

Apache Mahout Cookbook

（美）Piero Giacomelli　著
靳小波　译





机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Mahout 实践指南 /（美）贾科梅利（Giacomelli，P.）著；靳小波译．—北京：机械工业出版社，2014.6
（大数据技术丛书）
书名原文：Apache Mahout Cookbook

ISBN 978-7-111-46714-4

I. M⋯ II. ① 贾⋯ ② 靳⋯ III. ① 机器学习 ② 电子计算机–算法理论 IV. ① TP181 ② TP301.6

中国版本图书馆 CIP 数据核字（2014）第 100085 号

本书版权登记号：图字：01-2014-1083



Mahout 实践指南

[美] Piero Giacomelli 著

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：秦 健	责任校对：殷 虹
印 刷：	版 次：2014 年 6 月第 1 版第 1 次印刷
开 本：186mm×240mm 1/16	印 张：12
书 号：ISBN 978-7-111-46714-4	定 价：49.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

译 者 序

机器学习是人工智能领域里的一个重要分支，是进行复杂数据分析和构建智能系统的一个十分重要的研究方向。互联网数据不断地爆炸性增长标志着大数据时代的来临，机器学习领域在处理大规模数据时将面临新的挑战。研究者除了从软件方面发明新的时间复杂度更低的可扩展的算法之外，还应积极地从硬件架构方面进行改进，其中值得关注的方向有两个：将并行计算和分布式计算引入机器学习。

并行计算当前的热门方向是 GPU 计算，将传统上同时运行在多台机器上的任务交给单台机器上的图形处理器处理，这使得并行计算的费用大大降低。Shane Cook 撰写的《CUDA 并行程序设计：GPU 编程指南》[⊖]是这方面的经典参考书。GPU 的技巧已经大规模地应用到机器学习领域以改进传统的算法，两个有代表性的 GPU 机器学习库是 Theano[⊜]和 GPUMLib[⊜]。

分布式方面最有代表性的工作是 Apache Hadoop。它支持在大型集群中运行应用程序。最为重要的是，该架构是 Java 语言编写的开源软件框架，它实现了 Google 的 Map/Reduce 框架，可供商业或科研免费使用。Mahout 库就是在这样的背景下产生的。它建立在 Hadoop 的基础上，主要用于处理大规模的机器学习问题，其中核心算法有聚类、分类、协同过滤。同样，该库是开源免费的，且支持商业级别的机器学习方面的应用。

针对从事机器学习应用方面的开发人员以及机器学习理论研究方面的科研人员使用 Mahout，本书提供了非常有价值的参考。作者在将 Mahout 用于商业领域方面经验丰富，本书旨在降低 Mahout 初学者的入门门槛。本书特点如下：

- 通过分析大量的实例，展示了如何更好地使用 Mahout 算法，主要有分类算法、聚类算法以及遗传算法。
- 由浅入深讲解实例，帮助读者逐步掌握 Mahout 的应用方法。
- 图文并茂，让读者及时了解每一步操作之后的效果，帮助读者更好地检验学习进度。
- 写作方式独特，通过编码的方式帮助读者了解代码的目标及含义，避开代码背后复杂的机理。

⊖ 本书已由机械工业出版社引进出版，ISBN：978-7-111-44861-7。——编辑注

⊜ 参见http://deeplearning.net/software/theano/。——译者注

⊜ 参见http://gpumlib.sourceforge.net/。——译者注

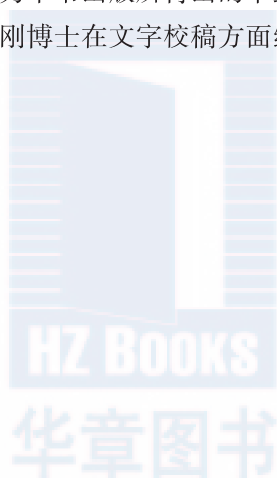
- 避开烦琐的数学表述，通过具体而形象的描述，让读者直观了解机器学习技术。

值得一提的是，Mahout 主要用在 Linux 平台上，但是对于使用 Windows 系统的大部分读者来说，这并不是一个障碍。本书通过详尽的描述，让不熟悉 Linux 的读者也可以学到 Linux 的基本使用技巧。实际上，本书中所有的代码都是在 Windows 系统下编写的，作者通过在 Windows 上安装 Virtual Box 软件来使用 Linux 平台，这种方式为那些 Windows 系统下的开发者使用 Mahout 库提供了一个良好的建议。

为了方便读者正确、迅速地理解本书，译者对本书的一些错误进行了修正，并在某些表述不太清楚的地方添加了注释，希望对读者理解本书内容有所帮助。然而，不得不承认，尽管译者从事的研究方向是机器学习，但由于水平有限，本书难免存在错误。欢迎读者及时向出版社指出，便于再版时予以更正。

特别感谢机械工业出版社编辑为本书出版所付出的辛勤劳动。

最后，感谢夫人靖莹以及耿光刚博士在文字校稿方面给予的支持和帮助。



靳小波

前　　言

在最近的 10 年，社会化网络的出现和移动设备的发明极大地改变了我们处理数据的方式。

为了帮助你了解究竟发生了什么，我们不得不提到在 2012 通过 Qmee 做的一项研究：在 60 秒里展示互联网上经常发生的事情。结果参考 http://blog.qmee.com/qmee-online-in-60-seconds/，它告诉我们在过去的每一秒里 Twitter 收到 278 000 条推文（tweet），Facebook 收到 41 000 条 post，YouTube 已上传了时长达 72 小时的视频。这些算是很大的网站了，但是即使是具有国家或国际背景的网站，因收集网站的日志而拥有上百万的记录也是很平常的。

为管理如此海量的信息，需要编写新框架来实现不同机器之间的计算任务共享。Hadoop 是 Apache 编写的算法解决方案，它可以将计算任务分配到不同的硬件架构上运行。

当你需要分析上亿条数据记录时，在大多数情况下，你的目的是通过信息提取来发现数据之间的新关系。传统上，数据挖掘算法就是为这个目的发展起来的。然而，当处理非常大的数据集时，无法在一个合理的时间里实现数据挖掘任务。Mahout 是一个数据挖掘框架，可以和 Hadoop 一起应用数据挖掘算法处理大规模数据集上的数据挖掘任务，它使用了封装在 Hadoop 里面的 MapReduce 例程。所以，Mahout 通过 Hadoop 架构的这个底层接口为编程人员进行数据挖掘任务提供了一个易用的框架。

本书将通过一些实例向你展示怎么使用 Mahout 进行数据挖掘，以及数据挖掘的不同方法。最关键的是，使用一种简洁通俗的方法向你展示使用 Mahout 对数据进行分类、聚类和预测的方式。本书是面向编程的，所以我们并不想在步骤中过多地引入其理论背景，但是我们会给有能力的读者一些参考文献以进行更深一步研究。当写这本书时，我们面临的主要挑战是：

- 根据我的经验，Mahout 有着非常高的学习曲线，主要原因是算法使用了 MapReduce 方法，该方法不同于序列方法。
- 数据挖掘算法本身就不太容易理解，它们在某些情况下需要一些特殊的技能，而开发人员不一定具备这样的技能。

于是我们尽力使用一种面向源码的方式让读者能抓住每一段代码的含义和目的，但是又不需要深入理解其背后的实现机制。

这种方法的效果由你来判断，而且我们希望你在阅读的时候能够发现一些乐趣，就像我

们在写作的时候获得的一样。

本书的组织

第 1 章描述如何在单台机器上创建一个完整的开发环境。通过编写一个推荐算法使得数据挖掘操作的所有代码片段均以 Hadoop 的方式呈现（包括引入的 JAR 库等），这非常清晰地展现在没有任何背景的读者面前。

第 2 章介绍序列文件。当使用 Hadoop 和 Mahout 时，序列文件是个比较关键的概念。在多数情况下，Mahout 并不直接操作要使用的数据集，所以在没有编码算法之前，我们需要描述如何对待这些特别的文件。

第 3 章详细介绍使用命令行工具和代码从 RDBMS 中读写数据。

第 4 章详细介绍如何使用朴素贝叶斯分类器分类文本文档。全面地描述如何将文档单词转化为包括单词出现次数的向量，并展示如何使用 Java 编写朴素贝叶斯分类器和互补朴素贝叶斯分类器。

第 5 章主要涉及两个算法：logistic 回归和随机森林（Random Forests）。它们展示了通过分析某些普通数据就可能预测其未来值。

第 6 章描述 Mahout 框架中最常用的算法，其中包括大数据的聚类分析和分类任务。在这一章，通过一些实例介绍使用顶棚聚类围绕聚类中心聚合数据向量。

第 7 章继续介绍 Mahout 中的聚类分析算法。该章描述了频谱聚类的使用方式，它在对图形式的链接信息进行分类时是非常有效的。

第 8 章描述了使用 K- 均值聚类（包括序列方式和 MapReduce 方式）对主题中的文本文档进行分类。我们将通过命令行方式和 Java 编码的方式解释如何使用该算法。

第 9 章介绍一个比较老的，称为频繁模式挖掘（Frequent Pattern Mining）的算法。该算法通过过去顾客的购买情况来预测哪些东西应该放在一块出售。Latent Dirichlet 算法将用于文本分类。

第 10 章描述了如何在 Mahout 中使用遗传算法解决旅行商（Travelling Salesman）问题和提取规则。我们将会看到如何使用 Mahout 的不同版本来使用这些算法。

阅读本书你需要什么

在第 1 章中，我们将介绍本书需要的所有软件。本书中所有的例子均在 Ubuntu 10.04 简易发行版和 Oracle 公司的 Virtual Box 平台上编程实现。

本书的读者

本书对希望以一种新颖、快速的方式来入门 Mahout 的开发人员是比较理想的。阅读本书不需要了解 Mahout，有经验的开发人员或系统管理人员也可以从本书中受益。

下载示例代码

你可以通过你的账号（在网站 http://www.packtpub.com）下载在 Packt 上购买的书籍的所有示例代码。如果你在别的地方购买本书，可以访问 http://www.packtpub.com/support 并注册，我们将会通过电子邮件直接把文件发送给你。

勘误表

尽管我们尽了最大的努力来确保内容的准确性，但错误在所难免。如果你在书中找到错误，无论在文中或代码中，我们会非常感谢你给我们报告了这些错误。你这么做，可以避免其他读者遭受困扰，并且我们将在随后的再版中改进。如果你发现任何错误，请通过访问 http://www.packtpub.com/submit-errata 来报告它们：选择你所购买的书名，点击“ errata submission form”链接，输入错误的详细细节。一旦你报告的错误得到确认，你的提交将会被接受，并且勘误表将会在我们的网站上更新或者加入已经存在的勘误列表中。通过网址 http://www.packtpub.com/support 选择书号，你可以看到现有的勘误条目。

关于评阅者

Nicolas Gapaillard 是 Java 架构方面的一位热情的自由撰稿人，他了解 Java 和开源领域中的创新项目。

他曾在开源软件公司 Linagora（http://www.linagora.com）的证券部门从事开发工作，由此开始他的职业生涯。该部门旨在开发一个围绕交易安全的开源软件，包括证书管理、密钥文档的存储和认证机制。

之后，他在 Smile 开源软件整合项目中任职 Java 技术方面的开发人员、培训人员和技术领导者。

有了上述从业经历之后，他决定创办自己的公司（名为 BIGAP,http://bigap.fr），该公司主要做自由撰稿业务，这使得他有更多的时间来学习和研究创新项目。

其中有一个业务是为名为 Onecub 的法国公司实现根据顾客的类别自动分类电子商务方面的电子邮件。当时，仅仅 Mahout 可以提供“拿来即用”的算法来解决这些问题。从那以后，Nicolas 开始深入地研究 Mahout 项目和数据挖掘领域。

某一天，Packt 出版社看到他撰写的文章（http://nigap.blogspot.fr）并邀请他为该书撰写评论，他非常愉快地接受了这项任务。

我非常感谢本书作者为保证书的质量而做出的努力，我也感谢 Packt 出版社，他们信任我，让我撰写该书的评论，他们非常仔细地管理整个流程，并且允许我评论该书的修订。我还想感谢其他的评论人为本书的修订和内容的质量而提供的帮助，感谢我的妻子让我有自由的时间来写这些评论。

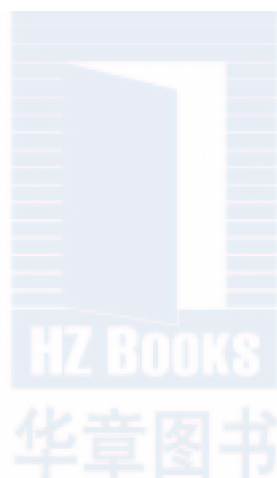
Vignesh Prajapati 是 Pingax 公司大数据方面的科学家。他热爱开源技术（比如 R 语言、Hadoop、MongoDB 和 Java 语言），主要工作就是使用机器学习、R 语言、RHadoop 和 MongoDB 进行数据分析。他在多个算法方面是专家，例如数据 ETL、电子商务、Google 历史分析和其他数据集的生成推荐、分析和行为定位等。他也撰写了几篇文章来阐述使用 R 语言、Hadoop 和机器学习实现高效的智能大数据应用。他的联系方式是 vignesh2066@gmail.com 或 http://in.linkedin.com/in/vigneshprajapati/。

除了本书以外，Packt 还有两本书与他有关：他是《Big Data Analytics with R and

Hadoop》一书的作者（Packt 出版，https://www.packtpub.com/big-data-analytics-with-r-and-hadoop/book），他也为《Data Manipulation with R》一书（作者：DeMystified，Packt 出版）撰写评论。

我感谢 Packt 出版社提供的这个难得的机会，感谢我的家庭、朋友和 Packt 出版团队激励和支持我为开源技术贡献自己的力量。

Shannon Quinn 将在 Carnegie Mellon 大学（匹兹堡）攻读计算生物方向的博士学位。他的研究兴趣是和他的导师 Chakra Chennubhotla 博士一起将谱图理论、机器视觉和模式识别用于生物图像识别，为生物监控构建实时分布式的框架。他还参与了 Apache Mahout 和其他开源项目的开发工作。



致　谢

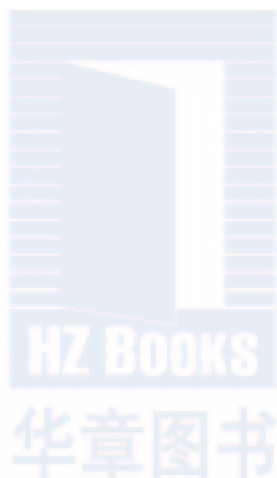
感谢我的家庭在我写书的最为紧张和激动人心的几个月里给予的支持。

感谢我的妻子 Michela，她每天激励我成为一个更好的人，而我母亲 Milena 在我结婚前就已经这样做了。另外，要感谢 Lia 和 Roberto，每次只要我们有所求 , 他们都会尽力帮助。

还要特别感谢整个 Packt 出版团队，感谢 Gaurav Thingalaya、Amit Singh、Venu Manthena、Shiksha Chaturvedi、Llewellyn F. Rozario、Amey Varangaonkar、Angel Jathanna 和 Abhijit Suvarna，他们对我给予了足够的耐心，尽管他们没有特别的原因要这么做。

最后，当我写这本书的时候，我找到一个新工作，这要感谢 Giuliano Bedeschi。他和他的两个儿子 Giovanni 和 Edoardo 一手创办了 SPAC 公司，这是我值得为之骄傲的为数不多的公司之一。在这个过渡期间，SPAC 公司的同事们真正地给予了我很大的帮助。

目　　录


HZ BOOKS
华章图书

第 1 章　Mahout 入门

Mahout 是一个机器学习 Java 类库的集合，用于完成各种各样的任务，如分类、评价性的聚类和模式挖掘等。

当前存在很多比较好的框架，它们对用户友好并且配置了更多的算法来完成这些任务。比如，R 社区比从前更加庞大，而在 Java 世界里，可用的 RapidMiner 和 Weka 库已经存在了好多年。

为什么我们要用 Mahout 来代替前面提到的那些框架呢？真正原因在于前面提到的那些框架并不是为大规模数据集设计的。当我们提到大规模数据集中所谓的数据集时，无论是哪种形式，它的记录都是上亿级别的。

事实上 Mahout 的威力在于这些算法可以用于 Hadoop 环境。Hadoop 是一个允许算法并行运行在多个机器上（称为节点）的一般框架，它使用了分布式计算的框架。

Hadoop 背后的核心理念不是使用单个的主节点来处理大数据的计算和存储任务，而是使用分治法将整个任务分成很多子任务。当所有的单个任务完成后（即每个任务完成计算并生成一个输出），Hadoop 将负责管理和重组所有的单个子集。在这种情况下，即使每个阶段并不是很强大，通过将繁重的计算任务分给许多单个计算节点机器来得到最后的结果仍然是可行的。这一理念和第一个分布式计算的例子（SETI@Home[㊀]和 Great Internet Mersenne Prime Search（GIMPS）[㊁]）非常接近，不同的是，我们实现的是分布式的机器学习算法。在本书中，我们将在各种各样实例的基础上以一种更好的方式来涵盖细节问题。

秘笈 1　安装 Java 和 Hadoop

本章的第一部分主要介绍在单台机器上设置工作环境，帮助读者以一种最容易和最快速的方式来编程。

就像之前说的那样，我们关注引导编程人员在开发机器上快速测试他们的 Mahout。我们不会过多地说明在一个产品式的 Hadoop 聚类上如何配置代码，因为那已经超出了本书的范围，需要更为详细和复杂的方法和配置。

我们仅仅需要让读者知道所有的工作都使用单个节点聚类，所以即使在不同的方法中，我们描述算法运行在多个聚类机器上所需要的参数时也是如此，在本书例子中，内部

㊀　参见http://setiathome.berkeley.edu/。
㊁　参见http://www.mersenne.org/。

计算经常被迫使用单个聚类。对于如何配置 Hadoop 聚类，请读者参考另一本书《 Hadoop Operations and Cluster Management Cookbook 》(Packt 出版，Shumin Guo 著)。

使用 cygwin 环境，在 Windows 系统上也可以测试 Hadoop 和 Mahout 代码，但是我们并没有包含这些内容，请读者参考 Apache 的相关 wiki：http://hadoop.apache.org。

考虑到 Hadoop 可以运行在云环境中，以测试为目的，有能力的读者可以使用 Amazon EC2 来设置单节点的 Hadoop 聚类。相关的配置资料可以在 Amazon EC2 wiki 上找到[一]。

当我们写本书的时候，微软发布了一个能运行在 Azure Cloud 上的 Hadoop 实现，但是我们没有测试它。你可以在网络上找到相关资料。

从 Cloudera 网站[二]下载 Hadoop（一个完全的安装版 Hadoop 系统的 64 位 Virtual Box）也是可行的。

不得不说，从头开始配置一个极小的系统将极大地帮助你理解 Hadoop 和 Mahout 是如何交互的。

有时，虽然不配置 Hadoop，Mahout 也可以用来测试代码。然而，事实上，因为只有利用 Hadoop 的性能和可扩展性 Mahout 才能发挥优势，我们将很少介绍这种方法。

因此，我们将 Hadoop 和 Mahout 安装到 Ubuntu 系统的 32 位机器上。为了创建快速的可复制的开发环境，我们将使用虚拟机方式。

我们更喜欢使用 VirtualBox 机器模拟器（开源软件）。如果对 VirtualBox 不熟悉，请参考书籍《 VirtualBox 3.1: Beginner's Guide 》(Packt 出版)[三]。

在本书中，因为我们使用了 Windows 7 专业版上的 VirtualBox 虚拟机并想从客户机上获得一个快速的应答，我们决定使用 Ubuntu 桌面版（10.04, 32 位）。考虑到主要使用基于 Debian 的命令，因此在基于 Debian 的分布上复制 Ubuntu 是可行的。

Hadoop 和 Mahout 不应该在根用户下运行[四]，所以我们创建了一个叫 hadoop-mahout 的用户来安装和做每件事情。

我们的安装策略是：

- 安装 JDK 1.7u9
- 安装 Maven 3.0.4
- 安装 Hadoop 0.23.5
- 安装 NetBeans 7.2.1
- 编译 Mahout 0.8-SNAPSHOT 源代码

读者也可以下载最新的 Mahout 二进制版本并包含必要的 jar 到样例工程中，但是使用

[一] 参见http://wiki.apache.org/hadoop/AmazonEC2。
[二] 参见http://www.cloudera.com。
[三] 参见http://www.packtpub.com/virtualbox-3-1-beginners-guide/book。
[四] 即登录时以根用户进入，但执行时不以根用户身份执行。——译者注

Maven 可以帮助读者更好地控制 Mahout 的发布和 JAR 包的版本。在这种情况下，所有的 Mahout 包依赖通过手工下载，这将是一个非常费时和乏味的工作。

Maven 也用于测试代码。我们将不会提及 Maven 所拥有的一些特征，读者可以参考 Packt 书籍[㊀]。

在进入编程步骤之前，需要安装所有的一切，首先从下载开始。

本章简要介绍如何创建一个单节点的 Hadoop 开发环境。

注意　建议读者仔细地读下一个秘笈，因为书中所有的其他秘笈都依赖于正确编译和运行的平台。

准备工作

首先下载 JDK，Hadoop 和 Mahout 需要 JDK 1.6 版本或更高版本，本书使用的是 JDK 1.7u9，可以从 Oracle 网站下载[㊁]。

Hadoop 也能够在虚拟机上的 OpenJDK 中运行，但是我们更喜欢使用 Oracle 的 JDK。

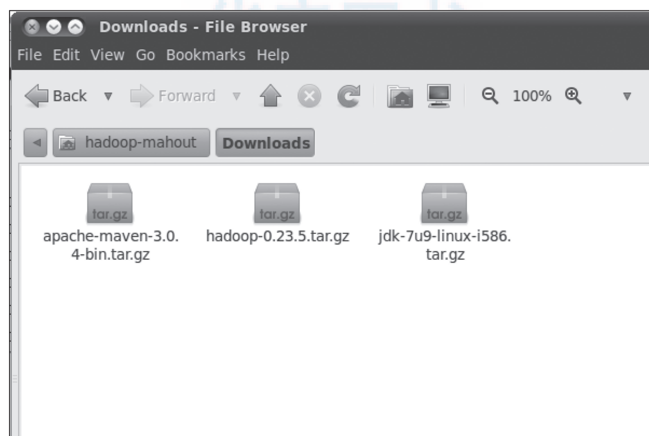
可以从 Apache 的某个镜像网站下载 Maven 3.0.4，下载终端命令如下：

```
wget http://it.apache.contactlab.it/maven/maven-3/3.0.4/binaries/apache-
maven-3.0.4-bin.tar.gz
```

然后，可以通过类似的方式下载 Hadoop 0.23.6：

```
wget http://apache.panu.it/hadoop/common/hadoop-0.23.6/hadoop-0.23.5.tar. gz
```

现在将下载的所有文件放到一个文件目录下，例如：/home/hadoopmahout/Downloads，你会看到下图所示的内容。



㊀ 参见http://www.packtpub.com/apache-maven-3-0-cookbook/book。

㊁ 参见http://www.oracle.com/technetwork/java/javase/downloads。

如何实现它

接下来将完成 Java、Maven 和 Hadoop 的环境设置。这三个框架的步骤几乎相同：

- ❑ 解压缩
- ❑ 添加正确的平台变量
- ❑ 测试安装的正确性

现在，我们将解压缩每个文档并将压缩结果从 /home/hadoop-mahout/Downloads 目录移到 /home/hadoop-mahout/ 目录。这是因为目录名称中的 Downloads 意味着它可以被别的软件使用或清除掉，而我们想保存我们的安装：

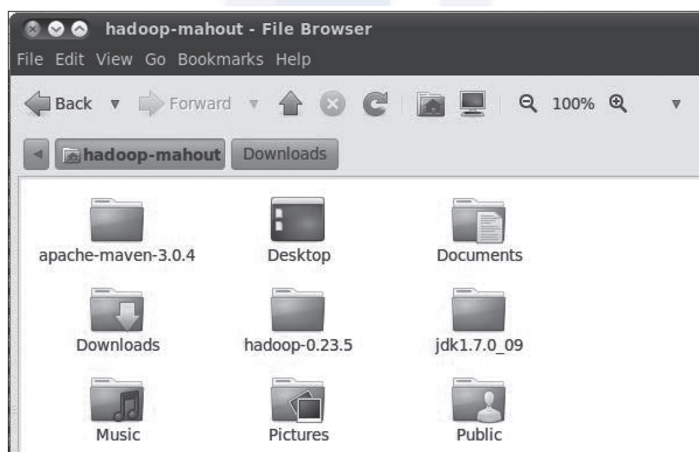
1. 在终端窗口输入下列命令进入 Downloads 目录：

```
cd /home/hadoop-mahout/Downloads
```

2. 输入下列命令：

```
tar -C /home/hadoop-mahout/ -xvzf jdk-7u9-linux-i586.tar.gz
tar -C /home/hadoop-mahout/ -xvzf apache-maven-3.0.4-bin.tar.gz
tar -C /home/hadoop-mahout/ -xvzf hadoop-0.23.5.tar.gz
```

3. 这将解压缩三个文档到目录 /home/hadoop-mahout/。hadoop-mahout 目录看起来如下图所示。



4. 为所需要的一切创建环境变量。这是因为 Maven、Hadoop 和 Mahout 需要一个配置变量 JAVA_HOME。

5. 我们也需要 Hadoop 能够从每个终端来访问 mvn 命令。为了让系统每次重启时都能使用这些变量，我们不妨把它们保存到文件 .bashrc 中。对单用户设置变量而言，这在 Ubuntu 上是一种容易的方式。

6. 为了完成所有这些设置，你需要：

- ❑ 用你喜欢的文本编辑器打开文件 .bashrc，它位于目录 /home/hadoopmahout/folder 下。

- 将下列代码输到文件末尾：

```
export JAVA_HOME=/home/hadoop-mahout/jdk1.7.0_09
export HADOOP_HOME=/home/hadoop-mahout/hadoop-0.23.5
export MAVEN_HOME=/home/hadoop-mahout/apache-maven-3.0.4
export PATH=$PATH:$JAVA_HOME/bin:$MAVEN_HOME/bin:$HADOOP_
HOME/bin
```

- 保存文件并返回控制台。

7. 前三行创建了用户变量：JAVA_HOME、MAVEN_HOME 和 HADOOP_HOME。
8. 最后一行将变量连同它们相对的 bin 位置加入 PATH 变量中。
9. 输入下列命令测试 JDK 和 Maven：

```
java -version
mvn -version
```

你将会得到如下图所示的输出。

```
hadoop-mahout@hadoop-mahout-laptop: ~
File  Edit  View  Terminal  Help
hadoop-mahout@hadoop-mahout-laptop:~$ java -version
java version "1.7.0_09"
Java(TM) SE Runtime Environment (build 1.7.0_09-b05)
Java HotSpot(TM) Client VM (build 23.5-b02, mixed mode)
hadoop-mahout@hadoop-mahout-laptop:~$
```

你也会看到如下图所示的输出。

```
hadoop-mahout@hadoop-mahout-laptop: ~
File  Edit  View  Terminal  Help
hadoop-mahout@hadoop-mahout-laptop:~$ mvn -version
Apache Maven 3.0.4 (r1232337; 2012-01-17 09:44:56+0100)
Maven home: /home/hadoop-mahout/apache-maven-3.0.4
Java version: 1.7.0_09, vendor: Oracle Corporation
Java home: /home/hadoop-mahout/jdk1.7.0_09/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "2.6.32-38-generic", arch: "i386"
hadoop-mahout@hadoop-mahout-laptop:~$
```

10. 为了测试 Hadoop 是否正确安装，我们将计算 Hadoop 发布的一个例子，它在一个单机上使用 10 个 MapReduce 工作集来计算圆周率 pi 的值。

11. 在 HADOOP_HOME 下输入下列命令：

```
hadoop jar /home/hadoop-mahout/hadoop-0.23.5/share/hadoop/
mapreduce/hadoop-mapreduce-examples-0.23.5.jar pi 10 100
```

12. 在终端你将会看到 10 个 MapReduce 工作集的工作情况：

```
Job Finished in 8.983 seconds
Estimated value of Pi is 3.14800000000000000000
```

到目前为止，我们已经成功地设置了一个单节点的 Hadoop 用于测试。我们现在开始下载并使用 SVN（Subversion）和 Maven 来编译 Mahout 源代码。

秘笈 2 设置 Maven 和 NetBeans 开发环境

这一节的内容比较基础，请仔细跟着做一遍，因为书中其他的设置依赖于 Maven 和 NetBeans 的成功安装。如果你觉得使用 Eclipse 比较舒服，建议你读读网站 http://maven.apache.org/eclipse-plugin.html 提供的帮助手册。然而，因为整本书都是基于 NetBeans 作为 IDE（可视化开发环境），一旦使用 Eplipse，每个配置都要重新检查。

准备工作

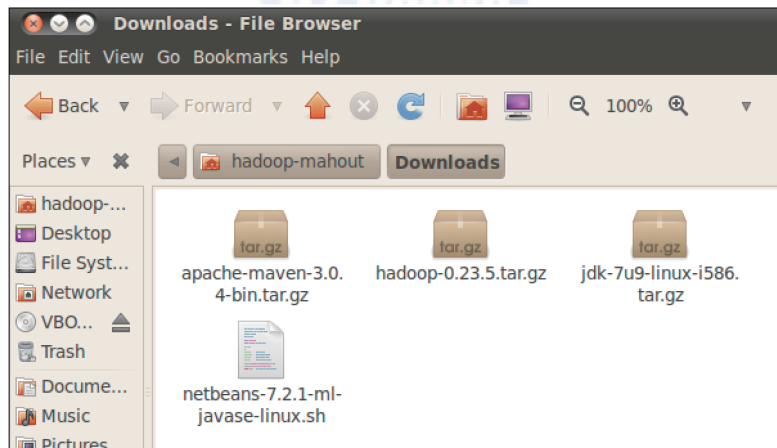
现在我们已经准备好进行最后的部分了——使用 NetBeans 安装和配置 Mahout。

出于编程的目的，我们决定使用 NetBeans 作为 IDE 代替其他的 IDE（比如 Eclipse），因为当写作本书时，Eclipse 的最新版本并不完全和 Maven 3.0.4 的规范兼容。使用 Eclipse 或 IntelliJ 也行，但是配置比 NetBeans 要麻烦些。如果想在 Elipse 上使用 Maven，可以参考：http://maven.apache.org/eclipse-plugin.html。

为了得到最新的发行版本，可以使用 SVN 编译最新的 Mahout 镜像，若有必要也可以下载二进制文件来链接 JAR 包。

可以通过 NetBeans 编译源文件，所以在进行下面的步骤之前，需要从 NetBeans 的网站[⊖]上下载文件 netbeans-7.2.1-ml-javase-linux.sh。

下载之后，Downloads 目录如下图所示。

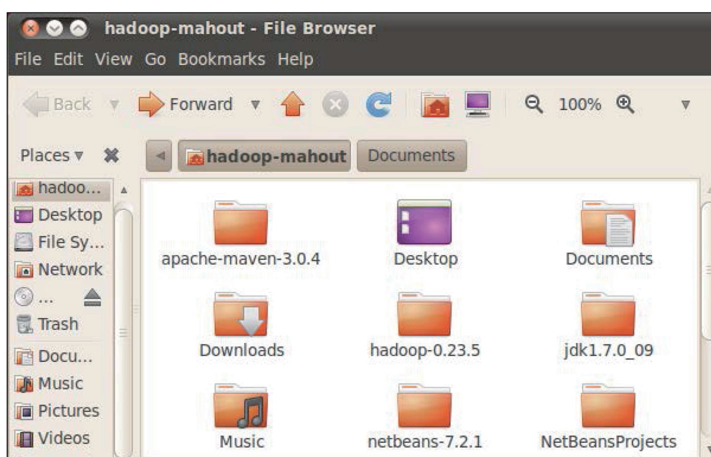


安装完 NetBeans 后，进入 Downloads 目录，然后在终端窗口输入下列命令：

```
/home/Hadoop-Mahout/Downloads/sh netbeans-7.2.1-ml-javase-linux.sh
```

执行该步骤后即可结束。hadoop-mahout 用户目录看起来如下图所示。

⊖ 参见www.NetBeans.org。



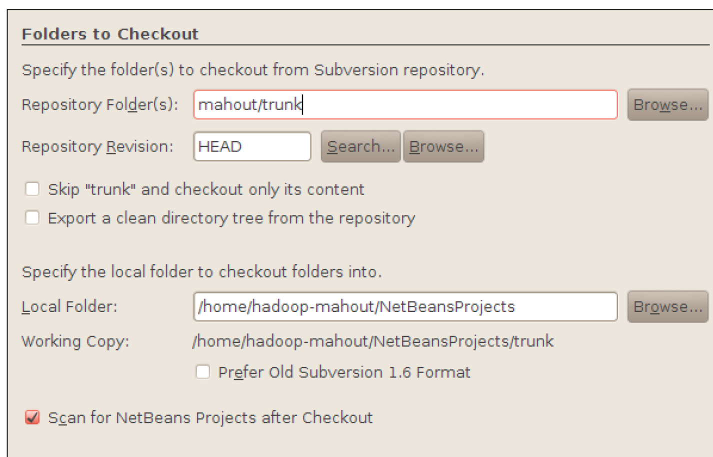
NetBeansProjects 目录包含 Mahout 源代码和新加入的代码。我们现在感兴趣的是实现在 NetBeans 上编译 Mahout 源代码。已经成功安装 NetBeans，接下来准备使用 NetBeans 编译 Mahout 的最新快照。

如何实现它

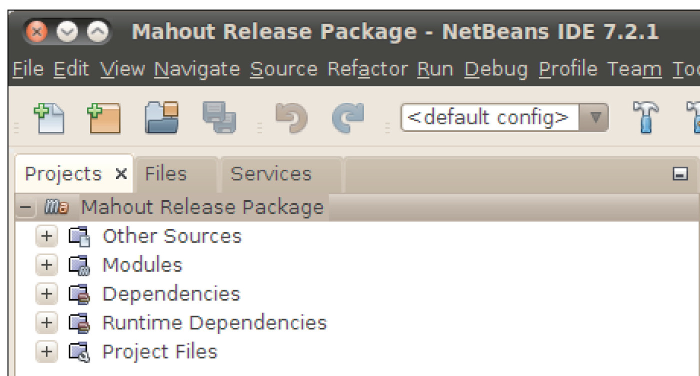
在写本书的时候，Mahout 的最新版本是 0.8。建议使用该版本，因为它去除了一些 bug，与此同时，新算法和新特征在被有活力的社区不断加入新的发行版本中。

1. 我们需要从 Subversion 下载 Mahout 源，将与 Maven 相关的工程导入 NetBeans，最后安装它们。

2. 幸运的是，NetBeans IDE 提供了所有这些操作，它们被整合到不同的图形界面上。简单地使用菜单，单击 Team → Subversion → Checkout，将 http://svn.apache.org/repos/asf/mahout/trunk 填入文本框——库网址中，单击 Next，填完表单如下图所示。



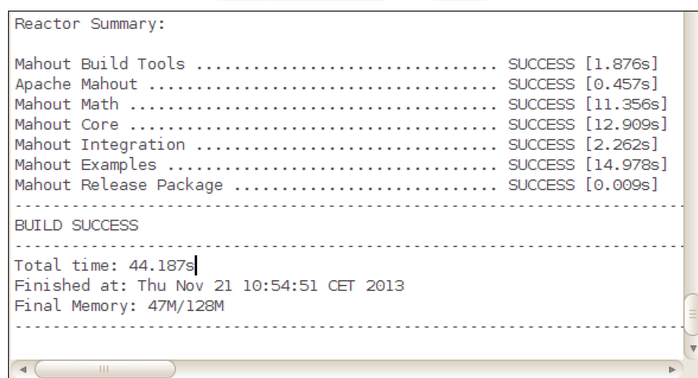
3. 一旦 NetBeans 下载完整个包，它会问你是否要扫描和打开那个工程。单击对话框上的 OK 按钮，IDE 将导入 Maven 工程。现在，在 Project 选项卡，你将会看到如下图所示内容。



4. 源代码下载完成后，可以在 NetBeansProjects 中找到它们。

5. 现在使用 Maven 编译这些源代码，在 Mahout Release Package 图标上右击，选择 Clean and Build 项即可。

6. 因为编译步骤需要较长的时间，此时可以稍作休息。当你结束休息时，将会看到如下图所示的输出。

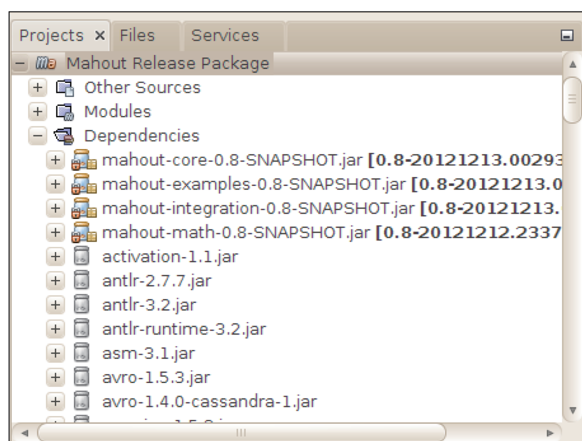


```
Reactor Summary:

Mahout Build Tools ............................... SUCCESS [1.876s]
Apache Mahout .................................... SUCCESS [0.457s]
Mahout Math ...................................... SUCCESS [11.356s]
Mahout Core ...................................... SUCCESS [12.909s]
Mahout Integration ............................... SUCCESS [2.262s]
Mahout Examples .................................. SUCCESS [14.978s]
Mahout Release Package ........................... SUCCESS [0.009s]
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time: 44.187s
Finished at: Thu Nov 21 10:54:51 CET 2013
Final Memory: 47M/128M
------------------------------------------------------------------------
```

我们现在有了测试 Mahout 所需的一切基础。

如果在 NetBeans 工程结构中单击依赖图标，你应该可以看到从 Apache 站点下载的所有 JAR 包及其依赖 JAR 包，如下图所示。



它是如何工作的

NetBeans 使用 Subversion 插件从 Mahout svn 库下载最新的源代码。一旦完成，NetBeans 将解析源代码库中的 pom.xml 文件，然后处理 Maven 源码并编译、测试它们，最后在目录结构下生成 JAR 包。

更多

NetBeans 提供给你的仅仅是操作 Subversion Maven 进程的接口。但是如果使用单一的文本编辑器编程对你来说有点困难，你可以直接使用命令行接口。

以相同的方式，你也可以使用 Eclipse 访问 svn 库并使用 Maven Eclipse 插件来编译它们。

不要忘记，一旦下载完源代码，在准备将 Maven 工程导入 Elipse 之前，你需要在源代码的根目录运行下列命令：

```
mvn eclipse:eclipse
```

这将创建一个可供 Eclipse 使用的工程导入文件。

注意　如果你没有遵循先前的步骤，你的代码可能会有版本问题或编译错误。

秘笈 3　编写一个基本的推荐系统

现在你已经有了一个完全配置好的 IDE，里面有从源代码编译的 Mahout 的最新发行版，我们终于可以测试第一个样例代码了。

为努力从用户的角度做到编码少一点得到多一点，我们将会体验一个使用 Mahout 编写推荐系统的例子。

顾名思义，一个推荐系统是一款软件，它能够根据你先前的偏好记录为你新的 (或存在的) 偏好提供一些建议。

比如，关于购买商品的推荐系统能够基于你先前的购买行为而建议下一次应该购买什么商品。

根据推荐算法所分析的数据类型的复杂性，存在不同类型的推荐算法。在我们的例子中，我们将使用 **Slope One 推荐算法**，它基于协同过滤（collaborative filter）方法。

自动推荐系统软件是数据挖掘历史上研究最早的问题之一。该问题大体上可以分两步解决：

- 读取大量数据，数据包含用户对某个条目的偏好值。
- 找到适合推荐给用户的那个条目。

每个在电子商务网站（比如亚马逊，Amazon）上购买过东西的顾客一定会看到站点向你推荐值得购买的新书或新东西。我们会采用用户提供的电影推荐数据，从中挖掘并发现其他还没有看过的电影，通过这种方式来模拟也会得到相同的结果。

准备工作

数据挖掘算法对数据是非常渴望的，你给的数据越多算法的输出就越精确。

在进入下一步之前，我们需要下载一些数据集供测试。我们将使用 GroupLens 数据集来实现电影推荐任务。

GroupLens 数据集是一个由明尼苏达州（Minnesota）大学计算机科学与工程系发布的可以自由获取的数据集，它包括 100 万条由 6000 个用户对 4000 个电影的评分等级。

数据可以以文本文件的格式读取，这对 Mahout 来说，是一种最简单的读取数据的方式。简单地在终端输入下列命令即可下载数据：

```
wget http://www.grouplens.org/system/files/ml-1m.zip
```

解压缩文件，你会看到目录包括下面四个主要的文件：

- users.dat：包含 6000 个用户。
- movies.dat：包含电影的名字。
- ratings.dat ：包含用户和电影之间的关系，该关系是一个数字，用来表示某个用户喜欢某个电影的程度。
- README ：主要是对数据格式的解释。

如果你打开 ratings.dat 文件（你马上就会用到），你会发现下面的数据行：

```
UserID::MovieID::Vote::datetime
1::1193::5::978300760
1::661::3::978302109
1::914::3::978301968
```

在每一行，你会看到一个电影评分，解释如下：用户 1 给电影《 One Flew Over the Cuckoo's Nest 》评了 5 分（最高 5 分）；给电影《 James and the Giant Peach 》和《 My Fair Lady 》评的都是 3 分。最后一个长的数字表示评分的日期 / 时间。

遗憾的是，尽管 Mahout 能够处理好输入是文本格式的情形，但这并不是 Mahout 容易使用的方式。这是因为在我们的例子里分隔符是“::”。这个简单示例将向你展示处理非标准数据格式时会遇到的一般问题。

在我们的例子里面，我们将原始问题转化为另一种格式，如下所示：

```
UseID,MovieID
1,1193
1,661
1,914
```

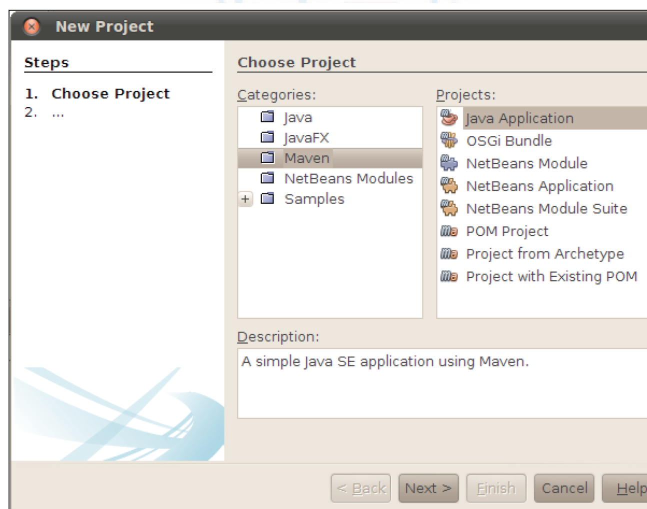
大体上，我们将一行行读取原始文件并拷贝每一行到新文件，在这个过程中，去除不重要的信息后就会得到最终的格式。

原始文件也包含对电影的评分值。推荐软件并不关心用户对电影的评分，于是我们删掉它们。我们也删除了评分的日期信息。

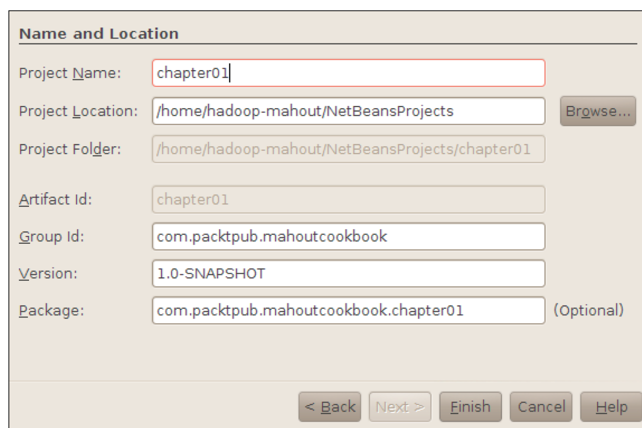
就如之前声明的那样，本书中都用 Maven 来创建例子，所以需要在 NetBeans 中创建 Maven 工程。具体步骤如下：

- ❑ 创建包括 main 类的 Maven 工程结构。
- ❑ 添加 Maven 依赖到先前编译的 Mahout Maven 工程。

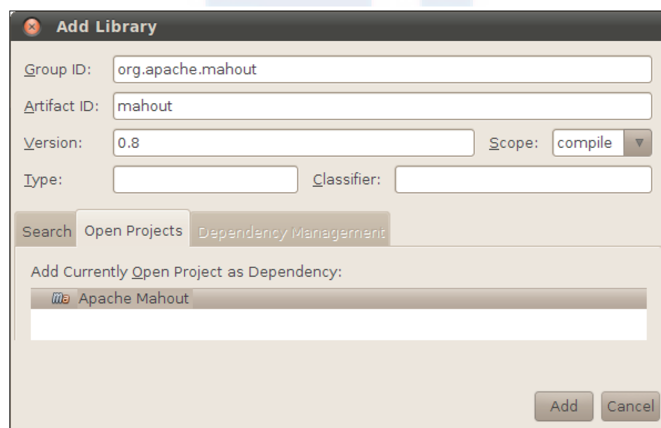
启动 NetBeans，打开“ File ”菜单选择 New Project，从显示的对话框中选择 Java Application，此时将会看到如下图所示内容。



完成之后如下图所示。



现在，我们需要添加包依赖到先前编译过的 Mahout Maven 源代码。为了在 Maven 工程目录结构下完成这件事情，右击依赖图标从快捷菜单中选择 Add dependency。选择依赖并添加完之后如下图所示。



单击 Add 按钮，过几秒，你将会看到加入的所有 Mahout JAR 包。

如何实现它

现在开始编写和测试第一个例子。步骤如下：

1. 将 GroupLens 格式的 ratings.dat 转化为 CSV 格式。
2. 创建一个 Model 类来处理将要使用的新文件 ratings.csv 的文件格式。
3. 在该模型上创建一个简单的推荐系统。
4. 从文件 ratings.csv 中提取整个用户列表需要花费一段时间，之后，在标题上将会看到

每个用户的推荐。

遵循前面的步骤，通过下面的代码将会完成它。在进入下一步之前，首先完成一些必要的导入操作。

1. 导入的包如下所示：

```
package com.packtpub.mahoutcookbook.chapter01;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

import org.apache.commons.cli2.OptionException;
import org.apache.mahout.cf.taste.common.TasteException;
import org.apache.mahout.cf.taste.impl.common.
LongPrimitiveIterator;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel
import org.apache.mahout.cf.taste.impl.recommender.
CachingRecommender;
import org.apache.mahout.cf.taste.impl.recommender.slopeone.
SlopeOneRecommender;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.recommender.RecommendedItem;

public class App
{
    static final String inputFile = "/home/hadoop-mahout/
Downloads/ml-1m/ratings.dat";
    static final String outputFile = "/home/hadoop-mahout/
Downloads/ml-1m/ratings.csv";
Moving to the main method and the core of our code we first code
a method to transform the original MovieLens file into a csv file
without the vote as explained before.

    public static void main( String[] args ) throws IOException,
TasteException, OptionException
    {
        CreateCsvRatingsFile();
```

方法的完整描述如下：

```
private static void CreateCsvRatingsFile() throws
FileNotFoundException, IOException
{
  BufferedReader br = new BufferedReader(new
FileReader(inputFile));

  BufferedWriter bw = new BufferedWriter(new
FileWriter(outputFile));

  String line = null;
  String line2write = null;
```

```
    String[] temp;
    int i = 0;
    while (
      (line = br.readLine()) != null
      &&
      i < 10000
      )
  {
    i++;
    temp = line.split("::");
    line2write = temp[0] + "," + temp[1];
    bw.write(line2write);
    bw.newLine();
    bw.flush();
    }
    br.close();
    bw.close();
  }
  }
```

2. 基于 CSV（Comma-Separated Value）文件建立模型，如下所示：

```
// create data source (model) - from the csv file
      File ratingsFile = new File(outputFile);
      DataModel model = new FileDataModel(ratingsFile);
```

3. 创建 SlopeRecommender：

```
  // create a simple recommender on our data
  CachingRecommender cachingRecommender = new
CachingRecommender(new SlopeOneRecommender(model));

        // for all users
for (LongPrimitiveIterator it = model.getUserIDs();
it.hasNext();)
{
  long userId = it.nextLong();
```

4. 最后，我们简单地显示推荐结果：[⊖]

```
  // get the recommendations for the user
  List<RecommendedItem> recommendations = cachingRecommender.
recommend(userId, 10);

  // if empty write something
  if (recommendations.size() == 0){
    System.out.print("User ");
    System.out.print(userId);
    System.out.println(": no recommendations");
  }
```

⊖ 读者在阅读这段代码的时候会有一个疑问，为什么作者会多出两个右括号呢？实际上，读第4步的代码应该连同前3步的代码一起看，其中对应的一个左括号在第3步的for循环处，另一个对应的左括号在第一步的类定义public classApp处。——译者注

```
    // print the list of recommendations for each
    for (RecommendedItem recommendedItem : recommendations) {
      System.out.print("User ");
      System.out.print(userId);
      System.out.print(": ");
      System.out.println(recommendedItem);
    }
    }
  }
```

让我们一起分析该代码看看它做了些什么工作。

下载示例代码　你可以通过网站 http://www.packtpub.com 上的账号下载你购买的所有 Packt 出版的书籍的例子代码。如果你在别的地方购买本书，你可以访问 http://www.packtpub.com/support 并注册，将会有一封电子邮件直接发给你。

它是如何工作的

根据该代码，我们有 import 语句。所有的导入包都来自于链接的 JAR 包，它们使用了 Mahout0.8 版本的依赖包。

第一个方法用作数据转换，即方法 CreateCsvRatingsFile。这段代码将原始文件转换为以逗号分隔的一个个的数值。如你所见，我们选择仅转换前 10 000 行。这样减少了算法运行过程中的计算时间并避免了在单台机器上可能出现的 Java 堆内存异常。

一旦这样做，我们将开始令人感兴趣的事情（包含在下面三行代码中）：

```
File ratingsFile = new File(outputFile);
DataModel model = new FileDataModel(ratingsFile);
CachingRecommender cachingRecommender = new CachingRecommender(new
  SlopeOneRecommender(model));
```

基于 CSV 文件创建完 DataModel 类之后，我们使用 SlopeOneRecommender 创建一个 CachingRecomender 对象。

Mahout 提供了各种不同的推荐算法，你也可以编写推荐算法。默认的推荐算法有：

- **基于用户的推荐算法（user-based recommender）**：该算法主要是通过相似度或邻居之间的距离来连接用户。
- **基于条目的推荐算法（item-based recommender）**：该算法在计算偏好值时并不仅仅使用近邻准则，还考虑将条目之间的相似度作为变量进行推荐。
- **Slope One 推荐算法**：它使用线性函数结合用户和条目来估计接近程度（proximity）。

在这个例子中，我们使用最简单且有效的一种算法——Slope One 推荐算法。

然后，我们遍历新文件 ratings.csv 中出现的每个用户，对每个用户，我们提取前 10 个推荐条目。

创建和编译之后，第一次运行会输出的推荐结果如下图所示。

```
Output - chapter01 (run) × App.java ×
User 1: RecommendedItem[item:3, value:1.0]
User 1: RecommendedItem[item:4, value:1.0]
User 1: RecommendedItem[item:5, value:1.0]
User 1: RecommendedItem[item:6, value:1.0]
User 1: RecommendedItem[item:7, value:1.0]
User 1: RecommendedItem[item:8, value:1.0]
User 1: RecommendedItem[item:9, value:1.0]
User 1: RecommendedItem[item:10, value:1.0]
User 1: RecommendedItem[item:11, value:1.0]
```

例如用户 1，推荐系统以 1.0 的概率向他推荐 ID 号为 3461 的电影⊖（对于像 Slope 这样的布尔值推荐算法而言，它输出的概率经常是 1）。

如果你查看一下 ID 号为 1 的用户资料，通过文件 users.dat 你会发现该用户是一名 18 岁以上的女人。对她而言，比如，根据文件 movies.data，被推荐的 3461 号电影是：

```
3461::Lord of the Flies (1963)::Adventure|Drama|Thriller
2::Jumanji (1995)::Adventure|Children's|Fantasy
```

令人感兴趣的是，这两部电影的情节都围绕着孩子的探险。即使电影《Lord of the Flies》的标题还可以更新，但考虑到用户的年龄，该推荐仍然给出了一个好的建议。

读者可以试着使用 100 个评分来代替 10 000 个，只要替换一行，以 i < 100 代替 i < 10 000 即可。

在这种情况下，我们观察到算法没有输出任何推荐结果，如下图所示。

```
Output - chapter01 (run) × App.java ×
run:
Nov 21, 2013 2:07:45 PM org.slf4j.impl.JCLLoggerAdapter info
INFO: Creating FileDataModel for file /mnt/new/ml-1m/ratings.csv
Nov 21, 2013 2:07:46 PM org.slf4j.impl.JCLLoggerAdapter info
INFO: Reading file info...
Nov 21, 2013 2:07:46 PM org.slf4j.impl.JCLLoggerAdapter info
INFO: Read lines: 100
Nov 21, 2013 2:07:46 PM org.slf4j.impl.JCLLoggerAdapter info
INFO: Building average diffs...
User 1: no recommendations
User 2: no recommendations
BUILD SUCCESSFUL (total time: 0 seconds)
```

于是结果明显地受 ratings.cvs 大小的影响，显然，如果你有更多的数据，那么推荐算法给你的推荐就更好。

另请参阅

现在我们已经编写了一个基本的推荐系统，在例子代码里，你也可以试着理解其他类型

⊖ 实际上我们从上述运行结果中看不出 ID 号为 3461 的条目。——译者注

的推荐算法。实际上，里面有一个立即可用的 GroupLens 推荐算法，它同样使用用户设置的偏好值。更为复杂的方法对有能力的读者理解 Mahout 推荐算法如何实现会更有用些。

了解推荐算法如何工作的一个更为正式的途径是参考下面这篇介绍性文章 http://dl.acm.org/citation.cfm?id=245121。如果想使用其他的数据集来测试推荐算法，你可以下载 Jester 数据集[1]。

关于与电子商务推荐的数据集，你可以去数据集超市[2]的网站看看。

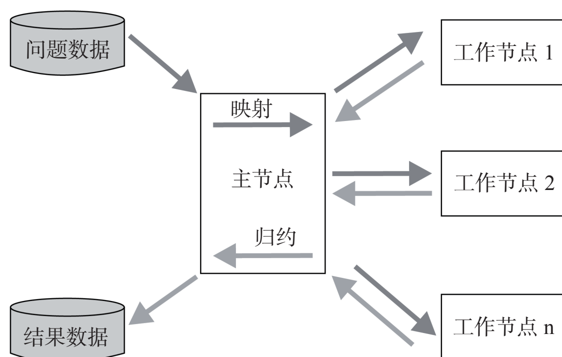
[1] 参见http://eigentaste.berkeley.edu/dataset/。

[2] 参见http://datamarket.com。

第 2 章　使用序列文件——什么时候和为什么

在上一章，我们简单地介绍了一下 Mahout 并通过一个完整的例子来演示如何使用 Mahout 编程。

从更高层次的角度来看，Hadoop 的 MapReduce 算法工作的示意图如下图所示。



从编写代码的角度看，我们有两个阶段：

- **映射（mapping）**：在这一阶段，原先的计算问题被分到主节点机器并被划分成更小的子问题。每个子问题被送到不同的工作节点（称为映射子，mapper）。
- **归约（reducing）**：在这一阶段，收集每个映射节点的输出被并将关键字索引相同的节点的输出组合在一起。

为了从概念上给出一个简单的例子，考虑 Hadoop 的 WordCount 类，它负责统计单个文本中单词出现的次数；输出是键 / 值对的集合，其中键是单词，值表示单词在文本中出现多少次。我们将文本划分成更小的部分，然后将每一部分分配给一个 mapper，mapper 记录单词在该部分的文本中出现的次数。最后，一旦每个 mapper 都完成工作，对重复出现的单词进行求和汇总就得到键 / 值对的整个集合。这种方法允许你分析非常大的数据集，唯一的限制是每个 mapper 都不应该超出它所在工作节点的内存容量。该框架的威力在于它能够并行地计算每个 map 工作，所以你可以让不同的工作节点同时工作在更小块的原始输入上。并行方法已被证明总体比序列方法快 5 倍，即使对于非常简单的算法比如归并排序也是这样。你可以看相关的文档，详见：https://www.vmware.com/files/pdf/VMW-Hadoop-Performance-vSphere5.pdf。

我们意识到这么大的数目处理起来一定要非常小心。这类方法在很多场景（包括理

论上和实际应用中）都取得了成功。尤其是，当数据集大幅增长时，这是唯一实际可行的方法。

然而，相比于传统的序列算法而言，并行算法的优势将会受限于下列事实：你不能使用同样的序列算法实现并行。这就是为什么我们对相同的算法考虑序列化版本和并行化版本时会有那么大的差别。随后我们会看到，好多机器学习和数据挖掘算法都是基于向量和矩阵的计算，这些实体很容易以并行化的方式整合在一起。

现在让我们转向本章的第一个秘笈。

秘笈 4 从命令行创建序列文件

在进入我们的具体方法之前，我们要有供测试用的数据集。我们选择使用 Lastfm 数据集。

准备工作

我们创建一个新的工作目录来开始工作。选择一个目录（在这个例子中是 /mnt/new/），输入下列命令：

```
mkdir  lastfm
mkdir ./lastfm/original
mkdir ./lastfm/sequencesfiles
export WORK_DIR=/mnt/new/lastfm
cd $WORK_DIR
```

于是我们创建了一个 lastfm 目录来存储想要使用的数据。为简单起见，我们设置了一个环境变量来存储绝对路径（在这个例子中是 /mnt/new/lastfm）。为了确保例子能运行你可以相应地改变它。

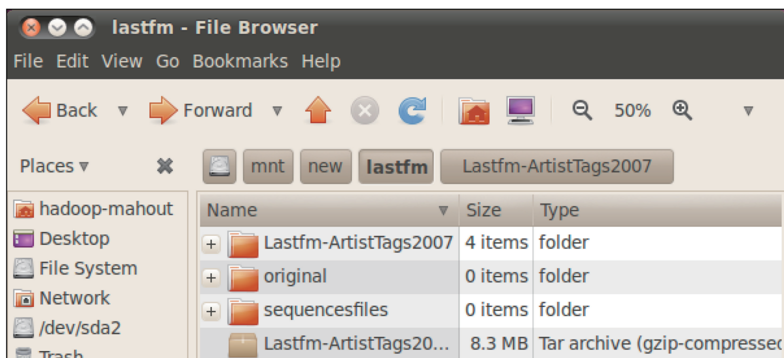
数据集 Lastfm 可以自由获取，使用下列命令可以下载它：

```
cd $WORK_DIR
hadoop-mahout@hadoop-mahout-laptop:/mnt/new/lastfm$ wget http://static.
echonest.com/Lastfm-ArtistTags2007.tar.gz
```

解压缩文件，使用下列的命令：

```
hadoop-mahout@hadoop-mahout-laptop:/mnt/new/lastfm$ tar -xvzf  Lastfm-
ArtistTags2007.tar.gz
```

现在在你的 $WORK_DIR 目录下应该有如下图所示的目录。



通过下列命令可以将原始文件复制到目录 $WORK_DIR/original 中：

```
hadoop-mahout@hadoop-mahout-laptop:/mnt/new/lastfm$ cp /mnt/new/lastfm/
Lastfm-ArtistTags2007/*.* /mnt/new/lastfm/original/
```

进入下一步之前我邀请你看看创建数据集的文件：

- ❑ Artists.txt：包含艺术家的注册信息。
- ❑ Tags.txt：包括数据集中的所有标记。
- ❑ ArtistTags.dat：列出了标记和艺术家之间的所有关系。

如何实现它

现在是时候将我们的第一个文件从它的原始格式转化为 Mahout 的序列格式了，其命令非常简单：

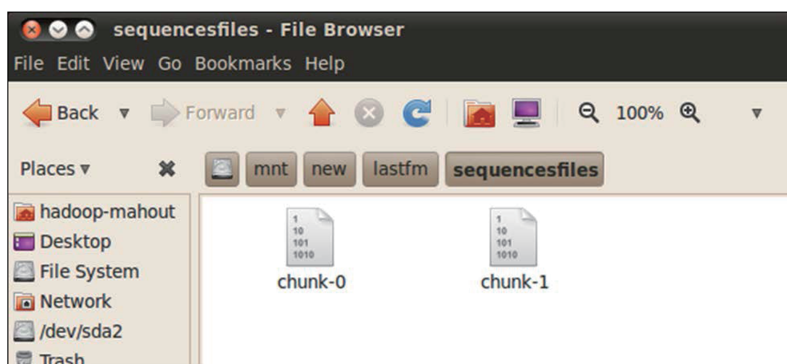
```
mahout seqdirectory -i $WORK_DIR/original -o $WORK_DIR/sequencesfiles
```

输出如下图所示。

```
Running on hadoop, using /home/hadoop-mahout/hadoop-1.0.4//bin/hadoop and H
MAHOUT-JOB: /home/hadoop-mahout/NetBeansProjects/mahout/examples/target/mah
Warning: $HADOOP_HOME is deprecated.

13/11/21 16:50:58 INFO common.AbstractJob: Command line arguments: {--chars
t/new/lastfm//original], --keyPrefix=[], --method=[mapreduce], --output=[/m
13/11/21 16:51:00 INFO common.HadoopUtil: Deleting /mnt/new/lastfm/sequence
13/11/21 16:51:00 INFO util.NativeCodeLoader: Loaded the native-hadoop libr
13/11/21 16:51:00 INFO input.FileInputFormat: Total input paths to process
13/11/21 16:51:00 WARN snappy.LoadSnappy: Snappy native library not loaded
13/11/21 16:51:01 INFO mapred.JobClient: Running job: job_local_0001
13/11/21 16:51:01 INFO util.ProcessTree: setsid exited with exit code 0
13/11/21 16:51:01 INFO mapred.Task:  Using ResourceCalculatorPlugin : org.a
13/11/21 16:51:01 INFO zlib.ZlibFactory: Successfully loaded & initialized
13/11/21 16:51:01 INFO compress.CodecPool: Got brand-new compressor
13/11/21 16:51:02 INFO mapred.JobClient:  map 0% reduce 0%
```

输出目录 $WORK_DIR 下的输出结果包括两个文件，如下图所示。



它是如何工作的

序列文件（sequence file）是键 / 值对的二进制编码。在文件的开始有一个文件头，其中包括一些元数据信息：

- ❑ 版本
- ❑ 键名称
- ❑ 值名称
- ❑ 压缩信息

可以使用下面的 seqdumper 命令来查看产生的文件：

```
mahout seqdumper -i $WORK_DIR/sequencesfiles/chunk-0 | more
```

输出如下信息：

```
Input Path: /mnt/new/lastfm/sequencesfiles/chunk-0
Key class: class org.apache.hadoop.io.Text Value Class: class org.apache.
hadoop.io.Text
Key: /tags.txt: Value: 440854 rock
343901 seen live
277747 indie
245259 alternative
184491 metal
158252 electronic
```

首先展示了 key 类和 value 类如何绑定到 Java 对象上（在这个例子中，两者都绑定到 org.apache.hadoop.io.Text 类）。

如果没有特别说明，解析默认使用普通的文本格式。

另一个有用的选项是 -ow 选项，它将会重写已存在的目标文件。

当解析文本文件时，选择 seqdirectory 非常有用，但考虑到不同的文件有不同格式，因此在这种情况下它没有什么意义。此外，当唯一定义键 / 值对的关系时，也可以得到所要求格式的文件。接下来转到代码部分看一看如何使用更为结构化的 Java 方法来创建序列文件。

秘笈 5　编写代码创建序列文件

在这个例子里面，我们将使用文件 Artists.txt 并利用文件中的唯一 ID 号、键 / 值对的名称来创建序列文件。原来文件的格式如下所示：

```
25231 Radiohead
20372 Pink Floyd
20251 The Beatles
19600 Red Hot Chili Peppers
18867 System of a Down
18671 Metallica
18671 Coldplay
18143 Nirvana
17629 Death Cab for Cutie
17507 Muse
16268 Green Day
16057 Franz Ferdinand
15306 Nine Inch Nails
15258 Led Zeppelin
15114 Tool
```

我们喜欢使用相同的格式来创建序列文件。

准备工作

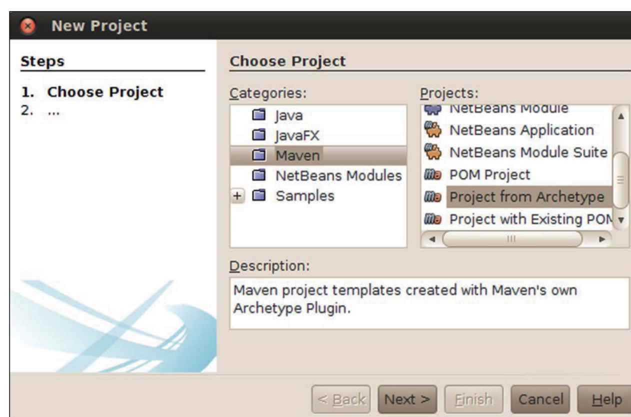
在转到目录 netbeansprojects 之前，打开终端并输入下列命令：

```
hadoop-mahout@hadoop-mahout-laptop:~/NetBeansProjects$ mvn
archetype:create -DarchetypeGroupId=org.apache.maven.archetypes -
DgroupId=com.packtpub.mahoutcookbook -DartifactId=chapter02
```

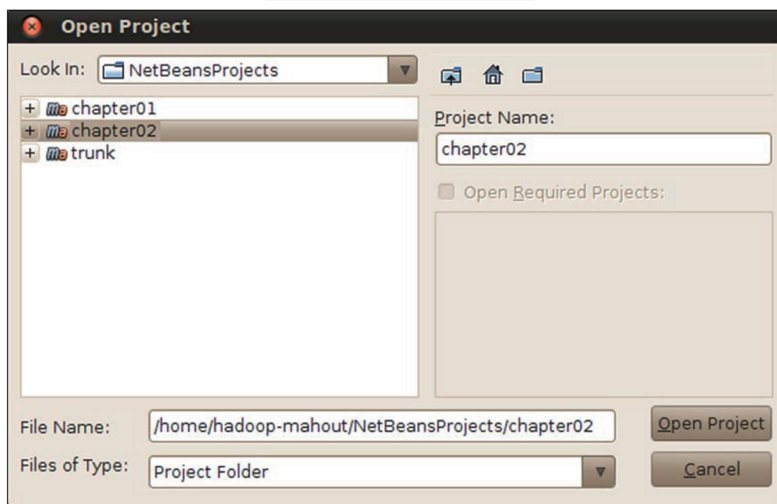
你将会看到如下图所示输出。

```
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Arche
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: com.packtpub.mahoutcookbook
[INFO] Parameter: packageName, Value: com.packtpub.mahoutcookbook
[INFO] Parameter: package, Value: com.packtpub.mahoutcookbook
[INFO] Parameter: artifactId, Value: chapter02
[INFO] Parameter: basedir, Value: /mnt/new/tmp
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /mnt/new/tmp/chapte
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 11.352s
[INFO] Finished at: Thu Nov 21 17:01:03 CET 2013
[INFO] Final Memory: 10M/24M
[INFO] ------------------------------------------------------------------------
hadoop-mahout@hadoop-mahout-laptop:/mnt/new/tmp$
```

打开 NetBeans，从 File 菜单中选择 New Project 就会出现如图所示的对话框。



下一步你将会看到由 mvn 命令创建的目录，如下图所示。

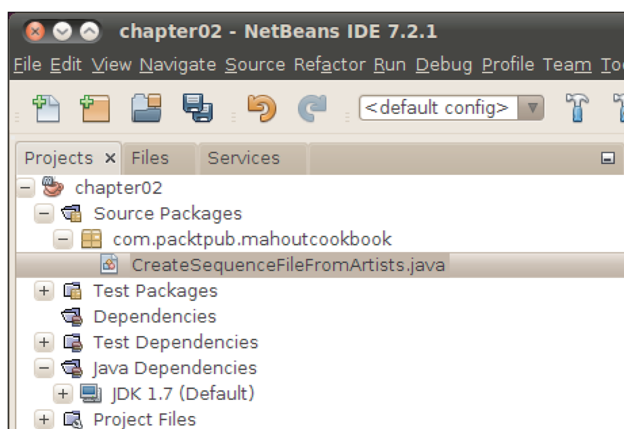


单击按钮 Open Project，导入的最终结果如下图所示。



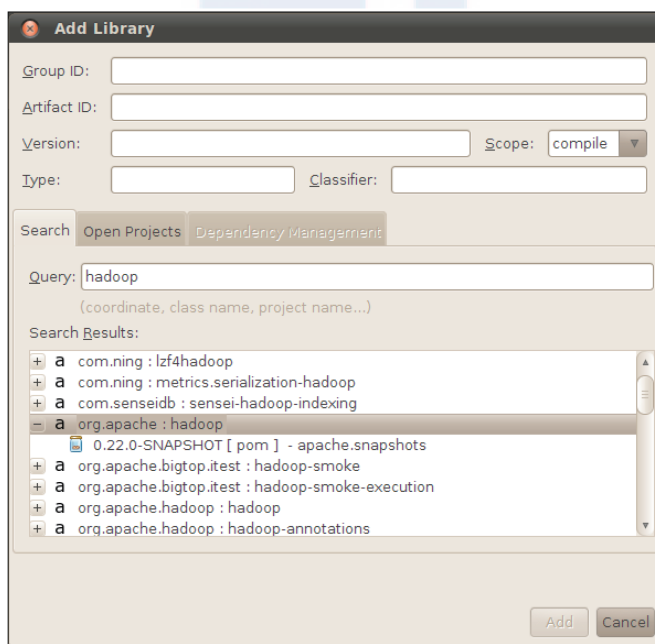
当使用 mvn 命令时，默认情况下会生成 App.Java 文件。你需要在 project 图标上右击，从快捷菜单中选择 Delete 来删除它。现在，你添加一个新的 CreateSequenceFileFromArtists 主类对象到工程中。

你将得到输出的如下图所示工程结构。

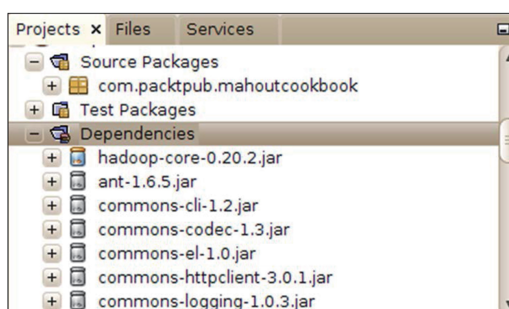


工程需要 Hadoop 才能运行，所以你需要链接包含 Hadoop 接口和类的 JAR 包，该包用在 mapreducer 中。我们创建的 Maven 工程能够使用在线的 Hadoop 库来下载所需的文件。

我们需要让每件事情都能正常工作才能加入 JAR 包依赖。为做到这件事情你需要右击 NetBeans 工程的 Dependencies 目录，从快捷菜单中选择 Add dependency，就会进入如下图所示对话框。



以同样方式加入 hadoop-core Maven 包来安装包依赖，如下图所示。



如何实现它

类 CreateSequenceFileFromArtists 的代码如下：

```
package com.packtpub.mahoutcookbook;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import org.apache.commons.beanutils.ConvertUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;

/**
 *
 * @author hadoop-mahout
 */
public class CreateSequenceFileFromArtists {
  public static void main(String[] argsx) throws
    FileNotFoundException, IOException
  {
    String filename = "/mnt/new/lastfm/original/artists.txt";
    String outputfilename =  "/mnt/new/lastfm/sequencesfiles/part-
      0000";
    Path path = new Path(outputfilename);

    //opening file
    BufferedReader br = new BufferedReader
      (new FileReader(filename));
    //creating Sequence Writer
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(conf);
    SequenceFile.Writer writer = new SequenceFile.Writer
      (fs,conf,path,LongWritable.class,Text.class);


    String line = br.readLine();
```

```
      String[] temp;
      String tempvalue = new String();
      String delimiter = " ";
     LongWritable key = new LongWritable();
     Text value = new Text();
     long tempkey = 0;
     while (line != null) {
       tempkey++;
       line = br.readLine();
       temp = line.split(delimiter);

       key = new LongWritable(tempkey);
       value = new Text();
       tempvalue = "";
       for (int i=1; i< temp.length;i++) {
         tempvalue +=  temp[i] + delimiter;
       }
       value = new Text(tempvalue);
       System.out.println("writing key/value  " + key.toString()
         + "/" + value.toString());
       writer.append(key,value);

     }

     writer.close();
     bf.close();

   }
 }
```

代码的输出如下图所示。

```
Output - chapter02 (run) ×
writing key/value  1/Pink Floyd
writing key/value  2/The Beatles
writing key/value  3/Red Hot Chili Peppers
writing key/value  4/System of a Down
writing key/value  5/Metallica
writing key/value  6/Coldplay
writing key/value  7/Nirvana
writing key/value  8/Death Cab for Cutie
writing key/value  9/Muse
writing key/value  10/Green Day
writing key/value  11/Franz Ferdinand
writing key/value  12/Nine Inch Nails
```


通过 hadoop 命令也可以得到相同的输出：

```
hadoop-mahout@hadoop-mahout-laptop:/mnt/new/lastfm/sequencesfiles$ hadoop
dfs -text part-0000
```

这与 Mahout 的 seqdumper 命令类似，但是它不需要目标输出文件。

输出显示如下图所示。

```
Warning: $HADOOP_HOME is deprecated.

13/11/22 13:19:18 INFO util.NativeCodeLoader: Loaded the native-hadoop libr
13/11/22 13:19:18 INFO zlib.ZlibFactory: Successfully loaded & initialized
13/11/22 13:19:18 INFO compress.CodecPool: Got brand-new decompressor
/tags.txt       440854 rock
343901 seen live
277747 indie
245259 alternative
184491 metal
158252 electronic
136691 punk
124599 pop
119930 indie rock
102937 classic rock
97264 alternative rock
89277 female vocalists
79497 emo
77455 death metal
```

它是如何工作的

如同我们先前所看到的那样，序列文件的格式包含键 / 值对。算法的主要执行步骤如下：

1. 打开文件 artist.txt 并一行行读取。
2. 对每一行使用一个计数器来为每个键创建唯一的索引。
3. 对每一行，读到那一行的 artist 域就创建一个 value 类。
4. 写键 / 值对到序列文件。

我们使用 BufferedReader 类对象打开文件。在下列代码中，创建 Sequence.writer 对象时有些棘手：

```
//creating Sequence Writer
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
SequenceFile.Writer writer = new SequenceFile.Writer
  (fs,conf,path,LongWritable.class,Text.class);
```

为创建序列文件，你需要声明 Hadoop 的 Configuration、FileSystem 类型和键 / 值对类。在这个例子中，我们使用预定义的 Hadoop 类——LongWritable 和 Text 类（对应 Java 中的 long 和 string 类型）。原始的文件 artist.txt 使用空格作为分隔符，我们需要划分每一行来找到艺术家的名称。

使用下列代码来添加键 / 值对：

```
writer.append(key,value);
```

最终，我们使用下列代码来关闭 Writer 对象：

```
write.close();
```

秘笈 6 编码实现读取序列文件

学习如何创建序列文件之后，是时候读取序列文件了。Mahout 提供读取序列文件的功能

并将每个键 / 值对转化成文本格式。命令行非常容易使用。比如，为了打印在前面的秘笈中创建的文件流，输入下列控制台命令即可：

```
mahout seqdumper -i $WORK_DIR/sequencesfiles/part-0000 -o
/mnt/new/lastfm/sequencesfiles/dump
```

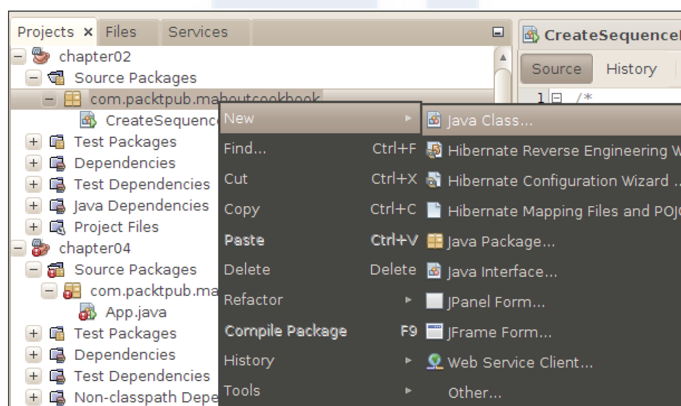
命令行将前面生成的 part-0000 文件的内容写到 $WORK_DIR 目录下的 dump 文件中。

然而，这种功能只能显示没有被使用的序列 seqdumper 的内容。

考虑到生成的序列文件将会被 Hadoop 的 mapper 和 reducer 使用和分析，我们将演示为了使用序列文件如何通过 Java 代码来读取序列文件。尤其是将读取一个序列文件并通过它创建一个 CSV 文件。

准备工作

为了能够运行，你需要在已有的 Maven 工程中添加一个新类。在之前创建的 NetBeans 工程中添加一个新的 Java 类，如下图所示。



当窗口出现时，在文本框中输入类的名称，在这个例子中是 ReadSequenceFileArtist，随后单击 Ok 按钮。

如何实现它

现在类已经准备好了，我们简单地通过下列步骤来添加一些代码到 main 方法中。

1. 首先，导入所需的类，如下列代码所示：

```
/*
 *
 */
package com.packtpub.mahoutcookbook;

import java.io.FileWriter;
import java.io.IOException;
```

```
import java.io.PrintWriter;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
```

2. 在 main 类中加入下列代码：

```
public class ReadSequenceFileArtist {

  public static void main(String[] argvs) throws IOException
  {
    String filename  = "/mnt/new/lastfm/
      sequencesfiles/part-0000";
    Path path = new Path(filename);

    String outputfilename =  "/mnt/new/lastfm/
      sequencesfiles/dump.csv";

    FileWriter writer = new FileWriter(outputfilename);
    PrintWriter pw = new PrintWriter(writer);
    String newline = System.getProperty("line.separator");
    //creating header
    pw.print("key,value" + newline);

    //creating Sequence Writer
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(conf);
    SequenceFile.Reader reader = new
      SequenceFile.Reader(fs,path,conf);

    LongWritable key = new LongWritable();
    Text value = new Text();

    while (reader.next(key, value)) {
      System.out.println( "reading key:" + key.toString() +
        " with value " + value.toString());
      pw.print(key.toString() + "," + value.toString() +
        newline);
    }
    reader.close();

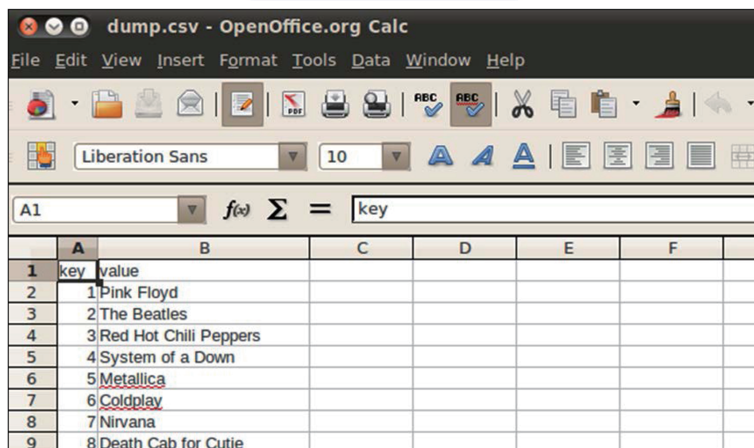
    pw.close();
    writer.close();


  }
}
```

3. 运行上述代码，输出目录将包含 dump.csv 文件，如下图所示。



4. 当使用 OpenOffice 打开文件时，可以看到如下图所示内容。



它是如何工作的

使用创建 PrintWriter 类的对象来处理输出非常容易。令人感兴趣的部分是创建 Sequence.Reader 类的对象。除了在读取键 / 值内容的情况下，reader 类的使用方式与 writer 类的是对称的。下列代码用来创建它：

```
//creating Sequence Writer
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
SequenceFile.Reader reader = new SequenceFile.Reader
  (fs,path,conf);

LongWritable key = new LongWritable();
Text value = new Text();
```

我们分别声明了两个对象来存储键类型和值类型。我们已经知道了它们是什么类型。如果不知道，现在你可以转化它的类类型。

我们使用 while 循环来读文件一直读到文件结束：

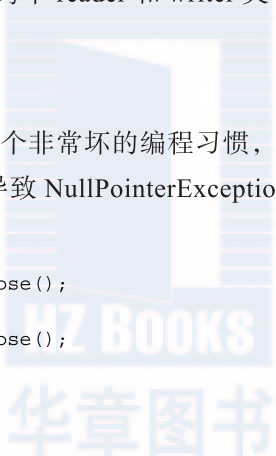
```
while (reader.next(key, value)) {
  System.out.println( "reading key:" + key.toString() + "
    with value " + value.toString());
  pw.print(key.toString() + "," + value.toString() + newline);
}
```

就像你看到的那样，每当调用 Sequence.Reader 对象的 next 方法时，就初始化一个新的键 / 值对象串。

在这个例子中，我们使用键 / 值对来生成一个新的 CSV 行，而 CSV 行以逗号隔开，在字符串的最后是系统的回车换行符。

最终，我们使用下列代码关闭每个 reader 和 writer 关联的对象：

```
reader.close();
pw.close();
writer.close();
```



按照前面的方式编写代码是一个非常坏的编程习惯，因为 writer 对象句柄可能由于某些原因会指向 null 句柄，程序将会导致 NullPointerException 错误。一种更好的方法是使用下面的代码：

```
if (reader != null) reader.close();
if (pw != null) pw.close();
if (writer != null) writer.close();
```

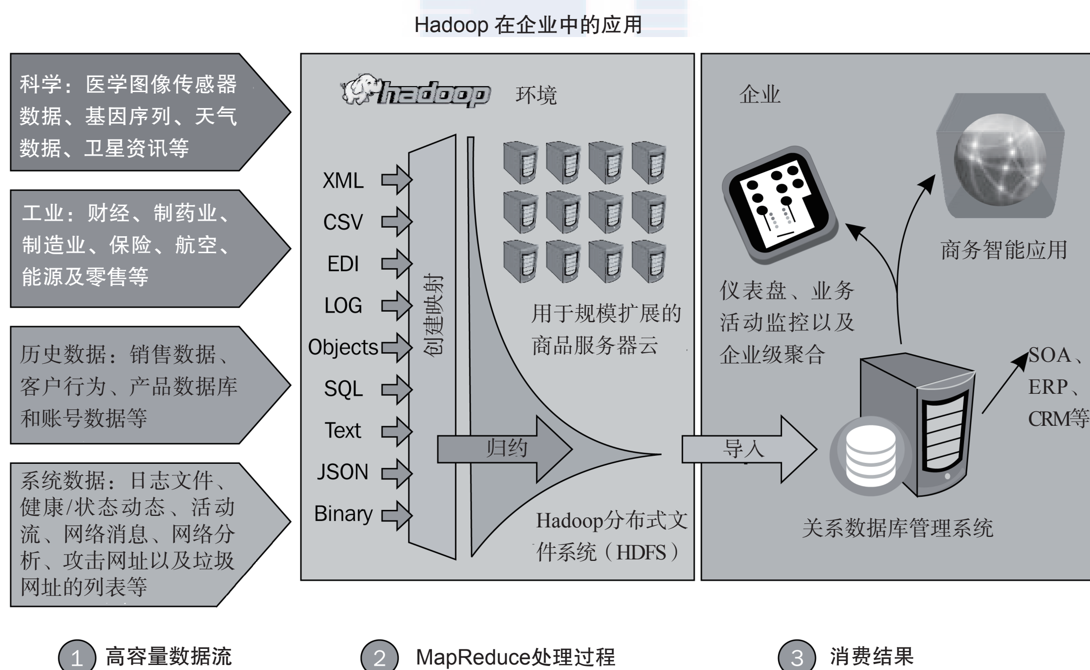
第 3 章　将 Mahout 和外部资源整合

到目前为止，我们已经知道了 Mahout 在单机环境下和分布式环境下分别是如何工作的。但是大体上我们仅仅能处理些文件，即由一些 MapReduce 作业生成的数据源文件或者序列文件。

然而，任何从事实际应用的编程者都知道除了一些嵌入式应用以外，90% 的数据都不存储在这类文件中。大多数情况下，数据以更加结构化的方式来存储。在大多数情况下数据存储软件将数据存储在关系数据库中，或者 NOSQL 数据库中，这是一种颇具潜力的新软件。

于是当我们为了数据挖掘读取一些数据时，我们将从关系数据库中读取它。在很多情况下，考虑到我们的 Mahout 分析也会产生数据，为了让其他的软件出于显示的目的能够读取它，我们需要以结构化表的方式来存储它。

根据我们在数据挖掘应用方面的经验，一个包含数据挖掘框架的结构良好的环境应该包含下图所示的架构。



为管理导入和导出，我们将使用 Sqoop。Sqoop 是另一个 Apache 软件基金项目，用于连接 Hadoop 生态系统与外部的数据资源和 RDBMS。

该工具是使用 Java 语言编写的，从算法的角度来看，它也是基于 MapReduce 框架的。如同目前所了解的那样，以并行的方式读取数据和在分布式的文件系统中使用它不同于以序列方式访问。这是因为当从 RDBMS 中读取一块数据之后，先前读取的数据块将会用于其他计算步骤，所以在开始计算步骤之前不必提取完数据集中的所有数据，然而这在序列式的 RDBMS 编程中却经常发生。

至于 Hadoop 平台的其他组件，该组件包含命令行工具和供 Hadoop 编程使用的 API。

首先介绍第一个例子，即导入数据到 HDFS（Hadoop 分布式文件系统）。

秘笈 7 导入外部资源到 HDFS

很显然，在开始之前我们需要创建一个 RDBMS 数据源来供我们测试使用。我们选用 MySQL 作为 RDBMS 系统，安装了一个测试数据库同时用作关于 HDFS 的读写和存储媒介。在我们的例子中，我们使用配置单 CPU 和 3GB 的内存的 VirtualBox Ubuntu 虚拟机作测试。我们安装的 MySQL 服务器版本显示如下图所示。

```
mysql> SHOW VARIABLES LIKE "%version%";
+-------------------------+-------------------------+
| Variable_name           | Value                   |
+-------------------------+-------------------------+
| protocol_version        | 10                      |
| version                 | 5.1.66-0ubuntu0.10.04.3 |
| version_comment         | (Ubuntu)                |
| version_compile_machine | i486                    |
| version_compile_os      | debian-linux-gnu        |
+-------------------------+-------------------------+
5 rows in set (0.00 sec)
```

我们已经在我们的机器上安装了测试数据库，但是在 Sqoop 被使用的 90% 的实际应用中，RDBMS 和 Sqoop 运行在不同的机器上。

开始测试之前的流程如下：

- ❑ 安装 MySQL 服务器。
- ❑ 导入测试数据库到 MySQL。
- ❑ 安装和配置 Sqoop。
- ❑ 格式化 HDFS。

准备工作

如我们先前看到的那样，我们需要安装一个工作用的 MySQL 服务器让客户端能够连接它，并在它上面安装一个测试数据库。完成这一步之后，我们需要为我们的第一个导入包安

装和配置 Sqoop。

为了在 Ubuntu 系统上做这件事情，我们打开一个终端并在非根账户（non-root account）下输入下列命令：

```
sudo apt-get install mysql-server
```

在安装的过程中，软件将会让你为数据库根账户设置密码，设置一个即可。我们将使用美国官方的棒球锦标赛统计数据的样本数据库来做测试，该数据来自网站 www.baseball-databank.org。执行下列一系列命令即可下载和安装该数据库[㊀]：

```
Wget http://www.baseball-databank.org/files/BDB-sql-2011-03-28.sql.zip
Unzip BDB-sql-2011-03-28.sql.zip
mysql -u root -p -e 'create schema bbdatabank;'
Mysql  -u  root -p  -s bbdatabank  < BDB-sql-2011-03-28.sql
```

当从命令行选项返回时，可以使用下列命令来检查一下安装后一切是否正常工作：

```
hadoop-mahout@hadoop-mahout-laptop:~$ mysql -u root -p -s bbdatabank -e
'select distinct name from Teams limit 10;'
Enter password:
name
Boston Red Stockings
Chicago White Stockings
Cleveland Forest Citys
Fort Wayne Kekiongas
New York Mutuals
Philadelphia Athletics
Rockford Forest Citys
Troy Haymakers
Washington Olympics
Baltimore Canaries
```

现在我们已经安装好数据库并准备使用它，我们可以开始安装 Sqoop 了。安装步骤相当容易，但是需要先前的那些设置，因为 Sqoop 使用 JDBC 驱动将自己绑定到 RDBMS。因此我们需要下载 Sqoop 和 mysql jdbc 驱动连接库。

为了进一步处理，读者应该打开一个终端，分别从 sqoop.apache.org 和 MySQL 网站下载 Sqoop 和 MySQL JDBC 驱动包（.jar 文件）。

重新开始之后，需要做下列事情：

1. 下载 Sqoop 和 mysql JAR 文件。
2. 解压缩 Sqoop 并将 JAR 文件拷贝到一个目录下。
3. 创建 SQOOP_HOME 环境变量。
4. 测试一切是否正常工作。

㊀ 文中的命令行有误：wget误写成Wget，unzip误写成Unzip，mysql误写成Mysql，但是Linux系统会区分大小写。——译者注

因为 Sqoop 只有在 Hadoop 上才能工作，所以需要基于先前的 Hadoop 安装来选择 Sqoop 的正确版本。在我们的例子中，使用的是 Sqoop 1.4.2 版本，它当前支持 Hadoop 1.x、0.20、0.23 和 2.0 版本。在任何情况下，在网站 http://www.apache.org/dist/sqoop/1.4.2/ 上，你都能看到与 Hadoop 兼容的 Sqoop 版本。下图显示了 1.4.2 版本的索引结构。

Index of /dist/sqoop/1.4.2

Name	Last modified
Parent Directory	
sqoop-1.4.2.bin__hadoop-0.20.tar.gz	2012-08-22 19:48
sqoop-1.4.2.bin__hadoop-0.20.tar.gz.asc	2012-08-22 19:48
sqoop-1.4.2.bin__hadoop-0.20.tar.gz.md5	2013-07-30 22:11
sqoop-1.4.2.bin__hadoop-0.23.tar.gz	2012-08-22 19:48

注意选择正确的 Hadoop 扩展文件。在我们的例子中，就如同在第 1 章中看到的那样，下载的版本是 Hadoop 0.23.5。在 Sqoop Apache 网址的镜像网站上可以获取 Sqoop 的二进制版本。可以使用下列命令得到所需的 Sqoop。

```
wget http://www.apache.org/dist/Sqoop/1.4.2/sqoop-1.4.2.bin__hadoop-0.23.
tar.gz
```

使用下列命令下载 mysql jdbc 驱动连接库：

```
wget http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-
java-5.1.22.tar.gz/from/http://cdn.mysql.com/
```

然后使用下列命令提取 Sqoop：

```
tar -xvzf sqoop-1.4.2.bin__hadoop-0.23.tar.gz
```

我们使用下列命令来提取 mysql JDBC 驱动连接库：

```
tar -xvzf mysql-connector-java-5.1.22.tar.gz
```

现在是时候配置和测试 Sqoop 了。就像我们之前对 Hadoop 和 Mahout 所做的那样，我们将使用 .bashrc 文件。用你喜欢的任何文本编辑器打开 .bashrc 文件并加入下列行到文件的末尾：

```
export SQOOP_HOME=/home/hadoop-mahout/sqoop-1.4.2.bin__hadoop-0.23
export PATH=$PATH:$SQOOP_HOME/bin
```

关闭当前的终端再打开另个一终端。为测试我们的 Sqoop 安装，输入下列命令：

```
sqoop /help
```

输出如下所示：

```
hadoop-mahout@hadoop-mahout-laptop:~$ sqoop help
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
```

```
Please set $HBASE_HOME to the root of your HBase installation.
usage: sqoop COMMAND [ARGS]

Available commands:
  codegen            Generate code to interact with database records
  create-hive-table  Import a table definition into Hive
  eval               Evaluate a SQL statement and display the results
  export             Export an HDFS directory to a database table
  help               List available commands
  import             Import a table from a database to HDFS
  import-all-tables  Import tables from a database to HDFS
  job                Work with saved jobs
  list-databases     List available databases on a server
  list-tables        List available tables in a database
  merge              Merge results of incremental imports
  metastore          Run a standalone Sqoop metastore
  version            Display version information
```

请参见 Sqoop 的帮助命令来查看关于特殊命令的一些信息。现在已经建立好了运行环境，接下来开始使用 Sqoop。

如何实现它

尽管读者会想 HBASE 是一个 Hadoop 分布式数据库，Sqoop 更希望使用它。但是，对于使用 Sqoop，HBase 并不是必要的，如果你想更好地理解 HBase 是如何工作的，我们建议你读读《HBase 管理员简明教程》(HBase Administration Cookbook，Yifeng Jiang 著，Packt 出版)。

不要弄混淆了：Sqoop 并不需要安装在导入数据 Hadoop 节点机器上。在这个例子中，我们从 RDBMS 系统到 Sqoop，再到 Hadoop 的一切都在同一台机器上；但是在真实的产品环境下，RDBMS 是另一个生态系统（ecosystem)，仅仅通过 IP 地址连接到 Sqoop 安装的机器上。

在产品环境下，你可能需要分配一个专门的机器给 Sqoop 用于制作预定的导入包。最后，同样重要的是，安装 Sqoop 之后，为能够和我们前面的章节创建的 mysql 数据库一起工作，我们需要添加 mySQL JAR 包文件。按下列步骤来做：

1. 简单地拷贝文件 mysql-connector-java-5.1.22-bin.jar 到目录 $SQOOP_HOME/lib 下。按照本书的设置，我们简单地输入下列命令：

```
cp /home/hadoop-mahout/Downloads/mysql-connector-java-5.1.22/
mysql-connector-java-5.1.22-bin.jar $SQOOP_HOME/lib
```

2. 前述的命令和下面的命令是等价的：

```
cp /home/hadoop-mahout/Downloads/mysql-connector-java-5.1.22/
mysql-connector-java-5.1.22-bin.jar /home/hadoop-mahout/sqoop-
1.4.2.bin__hadoop-0.20/lib
```

现在是时候测试导入并理解 Sqoop/Hadoop 和 Mahout 如何协同工作了。

我们简要地回想一下 Hadoop 的整个架构，Hadoop 是一个分布式文件系统，它在节点之间共享，共同地用作序列文件和文本文件的读写。在前面的一章，我们看到当你使用 MapReduce 作业时，你能够从 HDFS 中读写文件。

Hadoop 有它自己的数据库 HBase，用来存储并在 MapReduce 作业中共享数据，但事实上在 Hadoop 文件系统中没有什么是和 RDBMS 等价的。有经验的读者可能会问这样一个问题：我如何才能将数据（包括文件和 RDBMS 数据）移到 HDFS 中？

在移动的数据是文件的情况下，回答这个问题相当容易。Hadoop 自带了一系列的命令来模拟 Linux 终端上出现的基本命令。

比如，当你开始在 HDFS 上进行第一个测试时，比较好的一个方式就是格式化所用机器节点上的 HDFS。使用命令方式来完成非常简单，只需要打开一个终端窗口并输入下面的命令即可格式化：

```
hadoop namenode -format
```

在这个例子中，只有一个单节点配置，输出如下所示：

```
13/01/15 11:08:22 INFO namenode.FSNamesystem: supergroup=supergroup
13/01/15 11:08:22 INFO namenode.FSNamesystem: isPermissionEnabled=true
13/01/15 11:08:23 INFO namenode.NameNode: Caching file names occurring
more than 10 times
13/01/15 11:08:23 INFO namenode.NNStorage: Storage directory /tmp/hadoop-
hadoop-mahout/dfs/name has been successfully formatted.
13/01/15 11:08:24 INFO namenode.FSImage: Saving image file /tmp/hadoop-
hadoop-mahout/dfs/name/current/fsimage.ckpt_0000000000000000000 using no
compression
13/01/15 11:08:24 INFO namenode.FSImage: Image file of size 128 saved in
0 seconds.
13/01/15 11:08:24 INFO namenode.NNStorageRetentionManager: Going to
retain 1 images with txid >= 0
13/01/15 11:08:24 INFO util.ExitUtil: Exiting with status 0
13/01/15 11:08:24 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop-mahout-laptop/127.0.1.1
************************************************************/
```

注意 我们得提醒读者：我们不赞成在 Hadoop 0.22 中使用该功能。在那种情况下，使用的命令应该是 hdfs。比如，HDFS 的格式化命令应该是下面的格式：⊖

```
hdfs namenode -format
```

⊖ 前面的例子中第一个单词是hadoop，而这里第一个单词是hdfs，其他都一样。——译者注

HDFS 工具提供的各种命令的用法不在本章讨论的范围之内。我们建议那些想学习更多的有关命令用法的读者参考官方的 Hadoop 文档。我们需要澄清的是，当你使用 Hadoop 的 MapReduce 做序列文件或文本文件的读写操作时，所有的这些操作将存入 HDFS 中。

让我们开始我们的第一个导入操作。Sqoop 最初设计的时候就使用 JDBC 来连接数据库，所以就有可能把表和查询转化为不同的文件格式并将它们存入 HDFS 中。在这一节中，我们将会看到如何使用配置文件导入所有的 mysql 表。

在我们即将进行单个表或 SQL 查询的导入操作之前，让我们先试着导入前面存储在 bbdatabank 数据库（MySQL 形式）中的所有表。

做这件事情的控制台命令如下所示：

```
sqoop import-all-tables --connect jdbc:mysql://localhost/bbdatabank
--user root -P --verbose
```

在上述命令中最重要的参数是：

```
--connect jdbc:mysql://localhost/bbdatabank
```

上述的命令主要指示 Sqoop 在建立连接时应用使用哪个驱动。当触发 sqoop 时，包含驱动的 JAR 包应该在类路径上。这就是为什么将所有的 JAR 文件放在 $SQOOP_HOME/lib 目录下是一个好主意，在每次启动 Sqoop 脚本时默认将该目录加入类路径中。

连接属性的参数值以 JDBC 连接字符串的形式被写入。这暗示了每个支持 JDBC 连接的数据库都可以被 Sqoop 读取。然而，有经验的编程人员应该知道每个 DBMS 都有它自己的特性，所以任何新建的连接都应该被测试。考虑到还有其他的参数，在模拟 mysql 命令行时，我们有：

- --user：它表示了试着建立连接的那个 mysql 用户。
- -P ：它是运行时要求输入的密码。使用 --password 参数并在命令行中指明该参数并不是一个安全的做法，这是另外一种可能的做法。
- --verbose：它让我们能够使用全日志模式（full-logging mode），我们能看到它的输出，并且在任何情况下，我们都能够控制产生的异常。

我们可以把所有的东西存储在配置文件中，然后只需要传入配置文件的路径来调用 Sqoop，而不是调用一长行的参数。比如，上次调用所用的配置文件看起来应该是这样：

```
#
# Options file for Sqoop
#

# Specifies the tool being invoked
import-all-tables

# Connect parameter and value
```

```
--connect
jdbc:mysql://localhost/bbdatabank

# Username parameter and value
--username
root
-P
--verbose
```

通过传入配置文件来调用它应该是下面这样：

```
sqoop -options-file <path_to_file>
```

我们导入的最终结果在 HDFS 文件系统中可以使用下列命令浏览：

```
hadoop fs -ls
```

使用上面命令将显示下列文件的结果：

```
Found 25 items
-rw-rw-rw-   1 hadoop-mahout hadoop     601404 2013-01-15 14:33 TEAMS
-rw-rw-rw-   1 hadoop-mahout hadoop     601404 2013-01-15 14:33
ALLSTARFULL
-rw-rw-rw-   1 hadoop-mahout hadoop     601404 2013-01-15 14:33
APPEARANCES
-rw-rw-rw-   1 hadoop-mahout hadoop     601404 2013-01-15 14:33
AWARDSMANAGERS
-rw-rw-rw-   1 hadoop-mahout hadoop     601404 2013-01-15 14:33
AWARDSPLAYERS
-rw-rw-rw-   1 hadoop-mahout hadoop     601404 2013-01-15 14:33
```

针对这个例子，我们打开 TEAMS 文件，使用的命令如下：

```
hadoop -fs tail TEAMS
```

结果如下：

```
,767,668,4.24,1,14,40,4310,1409,154,605,1268,144,135,0.980,"Chicago
Cubs","Wrigley Field",\N,108,108,"CHC","CHN","CHN"
2010,"NL","CIN","CIN","C",1,162,\N,91,71,"Y","N","N",
"N",790,5579,1515,293,30,188,522,1218,93,43,68,50,685,648,4.02,4,9,43,
4359,1404,158,524,1130,86,140,0.988,"Cincinnati Reds","Great American
Ball Park",\N,99,99,"CIN","CIN","CIN"
```

该格式是 CSV 格式（属性值以逗号隔开）。

它是如何工作的

在 verbose 模式下，来自标准输出的导入步骤是非常简单的。每当你执行导入操作时（导入所有的任务或单个导入），Sqoop 都使用类路径中指明的 driver 类来连接数据库。这是我们为什么把它放在 $SQOOP_HOME/lib 目录下，所有的 JAR 文件都会被自动地加载。根

据 import 参数，Sqoop 将在关系数据库和目标地址之间生成一些映射类，最后使用该映射将结果转换为最终所要求的格式。

更多

在这个例子中，我们使用一个非常简单的方法只是想让读者了解 Hadoop 环境下 Sqoop 的重要性。但是更高级别的数据分析方法只会导入必要的数据来供 Mahout 分析使用。

Sqoop 也提供了基于 SQL 语句导入数据的可能性。我们将执行下面两个操作来测试它：

- 创建 Sqoop 配置文件。
- 使用免费的 SQL 语句来运行 Sqoop 导入操作。

我们利用这项技术让读者能够在相同数据库上测试不同的查询，但又不必重复输入它们。打开文本编辑器，为 Sqoop 创建下列连接属性文件：

```
#
# Options file for Sqoop
#

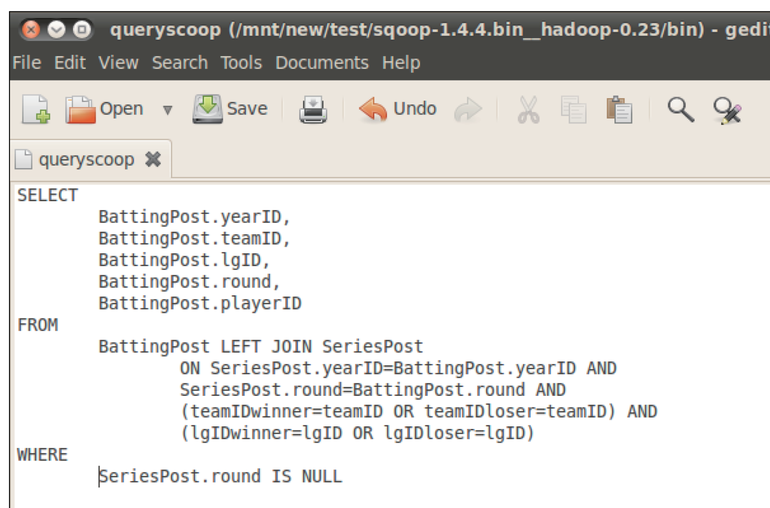
# Specifies the tool being invoked
import

# Connect parameter and value
--connect
jdbc:mysql://localhost/bbdatabank

# Username parameter and value
--username
root
-P
--verbose
```

将文件存为 sqoop.config 并把它放在 SQOOP_HOME 下（在我们的例子中，它对应 /home/hadoop-mahout/sqoop-1.4.2.bin__hadoop-0.20）。

你可能会注意到，相对于以前的配置，一个重要的变化是我们指定了 import 命令，于是我们不用导入每个表，而是仅仅导入包含数据的一个子集。这与调用 sqoop import 命令行的方式等价。运行 MySQL 语句显示如下图所示：



这条语句返回 BattingPost 中的所有的队员，这些队员出现在 SeriesPost 所表示的球队中。把我们已经有的这两部分放在一块并将查询的结果以 CSV 文件的形式导入 HDFS 中，我们调用下列命令：

```
sqoop  --options-file /home/hadoop-mahout/sqoop-1.4.2.bin__hadoop-0.20/
sqoop.config -query 'SELECT BattingPost.yearID, BattingPost.teamID,
BattingPost.lgID,  BattingPost.round, BattingPost.playerID FROM
BattingPost LEFT JOIN SeriesPost ON SeriesPost.yearID=BattingPost.
yearID AND SeriesPost.round=BattingPost.round AND  (teamIDwinner=teamID
OR teamIDloser=teamID) AND (lgIDwinner=lgID OR lgIDloser=lgID)  WHERE
SeriesPost.round IS NULL'
```

输出将是一系列的普通的 CSV 文本文件，它们存储在 HDFS 文件系统中并可以供其他的 MapReduce 作业使用。

我们建议读者参考 sqoop.apache.org 上由 Sqoop 提供的一些好的文档。在实际导入的过程中，我们还将指出其他一些对你有用的命令行参数，我们把这些参数列在这里：

- --as-sequencefile
- -m 和 --num-mappers
- --target-dir

我们来一个个地讨论，来自查询的 --as-sequencefile 参数强制文件以 Hadoop 序列文件键 / 值对的格式存储。在这种情况下，虽然所有的结果集都是键所对应的值，但是用于 Hadoop 计算的键值必须是唯一的并且会自动地生成。

Sqoop 使用 MapReduce 算法提供的并行化方式来从 RDBMS 导入数据。默认情况下，如果不指定参数，4 个 mapper 都用来查询 RDBMS。但是在应对包含上百万条记录的大型数据库时，你可以使用更多的 mapper。注意，你可以任意地设置这个参数，因为每个 mapper 都会打开一个专用的 RDBMS 连接。比如，如果你指定 100 作为并行 mapper 的参数，那么你

打开连接的数目将增加到 100 个，这将会对其他的已经连接的会话（connected session）引起某些性能方面的问题。

所以当你运行并行任务时，Sqoop mapper 应该能够识别出某个列，它可以用于划分被导入的数据。比如，如果你决定导入一个包含 100 万记录的表（数据库表）并使用四个 mapper 来做这件事情，Sqoop 需要唯一地确定如何将 250 000 条记录分配给每个 mapper。默认情况下，Sqoop 将识别主键列来做这件事情。然而，在普通的 SQL 语句中，可能不存在主键列，所以我们需要使用 --split-by 参数来指定要使用哪一列，该列将用于在不同的 mapper 之间划分工作量。

通过最后一个参数，我们观察到 HDFS 是在模拟 Linux 目录结构；所以在大部分时间里你需要一个工作用的导入目录。在这种情况下，你可以使用 -target-dir 参数，它让你能够指定 HDFS 中的目录，你需要将导入的结果文件放在该目录下。

秘笈 8　将数据从 HDFS 导入到 RDBMS

如同我们在前言中谈到的那样，我们将一个 Hadoop/Mahout 的挖掘过程分为三个主要的步骤：

- 将提取的数据导入 HDFS 实例中。
- 在 Hadoop/Mahout 上进行计算任务。
- 将结果导出到另一个 RDBMS 中，某个第三方软件将负责显示它。

现在我们已经看到了导入部分，我们需要提供导出的一些例子。按照命令行的风格，我们将使用 export 参数来调用 Scoop 主脚本。

我们不需要任何其他的配置，我们前面设置的所有东西对于导出目的仍然是有效的。

如何实现它

让我们从一个基本的例子开始。想象一下，在我们的 HDFS 里面有一个 CSV 文件叫 export.csv，它位于 /export/ 目录下。我们想把这个文件保存在一个叫 results 的 MySQL 数据表中。

使用下面的命令可以完成它：

```
sqoop export --connect jdbc:mysql://localhost/bbdatabank --user root -P
--verbose --export-dir /export/ --table results
```

从上面的命令可以看出，一旦设定参数来做这个工作，在 export 命令中，我们让这些可选参数简单地首尾相连即可。但是作为我们的第一个例子，我们需要给读者一些说明。

它是如何工作的

因为目标文件是 RDBMS 表，我们提醒读者目标表必须在目标数据库中存在。Sqoop 并不能自行地创建目标表。如果没有其他的指定，Sqoop 将创建一个 INSERT INTO sql 语句的集合，这些语句将在目标表上执行。当运行 Sqoop 的 export 命令时，需要强制指定的参数有：

- --export-dir
- --table

如果你先前有通过 Java 和 JDBC 使用 SQL 语句的经验，你一定会意识到在一个普通表上执行 insert 语句会有一些潜在的问题产生。我们总结一下这些潜在的问题，把它们列在这里：

- 两次插入同一个值导致的重复插入问题。
- 插入一个空值到必需的或非空值域列。

作为应对，Sqoop 提供了更多的容易调整的参数来避免这样的问题。让我们来分别地检验一下。

回到我们谈论的潜在问题，让我们看看如何管理重复的 insert 语句。默认情况下，Sqoop 仅仅执行一个 insert 语句，所以万一有两个源文件包含相同的记录，它将创建一个包含重复键的 insert 语句，所以为了避免潜在的运行期异常，你可以使用 --update-key 参数。该参数让你定义你目标表上的主键列，它可以用来进行更新而不能创建插入。

举个例子，我们有一个 CSV 源文件，它的行结构如下：

```
Id, movie, score
1,1,0.1
2,1,10
```

我们使用 mysql 命令创建表，其结果如下：

```
Create table results
(
Id int primary key not null
,movie int not null
,score float not null
)
```

如果每件事情都可以顺利进行，应该不会有任何问题，但是万一再有一个 ID 为 1 的记录，你的导出可能会产生重复键异常。你有两种可能的解决方法。第一种是使用 --update-key 参数并执行下列的 Sqoop 命令：

```
sqoop export --connect jdbc:mysql://localhost/bbdatabank --user root -P
--verbose --export-dir /export/ --table results --update-key id
```

当做插入操作时，上述命令强制 Sqoop 产生一个类似于下面的 SQL 命令：

```
update result set movie=<movie_value>, score=<score_value> where id = ..
```

如果 ID 号没有在目标表中出现，即使在你第一次导入 ID 或者添加一个新 ID（相比前一个）时没有出现错误，这还是没有完全解决问题。为正确地构建一个工作流，你应该首先导

出数据去填充表，再导出另一个带更新键的数据来避免重复键异常。

在这种环境下，管理插入和更新语句的最有用的参数可能就是带 --update-key 的 --update-mode 了。为了澄清这件事件，让我们考虑下面的 Sqoop export 命令：

```
sqoop export --connect jdbc:mysql://localhost/bbdatabank --user root -P
--verbose --export-dir /export/ --table results --update-key id -update-
mode allowinsert
```

在这种情况下，通过指定 allowinsert 模式，导入流将遵循下列逻辑：

- 检查更新键参数的值是否在目标表中存在。
- 如果是的话，创建并使用更新命令来更新信息。
- 否则创建一个 insert 语句。

这将会避免所有潜在的情况，但是记住一旦你开始使用更新语句，仅仅最后一次更新将得到最终的值。

当处理 insert 和 update 时，需要仔细考虑数据表上更新的域值。比如，如果你允许目标表中的 string 域接受一个空值，那么你不得不在对应的数据源中管理空字符串。Sqoop export 命令另外提供了一些参数来管理输入的分隔符和空值，再一次提醒读者参考 Sqoop 网站上的完整文档。我们仅仅引用其中会用到的两个参数：

- --input-fields-terminated-by ：如果指定该参数，它将说明输入文件中哪个字符用作域分隔符，默认情况下是逗号。
- --input-lines-terminated-by：该参数指定行分隔符，默认值是 \r。

考虑到空值，我们有：

- --input-null-string ：如果指定该参数，它将向导出工具指明哪个字符串将被认为是空值。注意，默认情况下，null 被解释为空字符串。
- --input-null-non-string ：如果该参数没有指明（默认情况下），在 SQL 语句中，空字符串和空值都会被认为是 null。

现在我们已经看到两种数据流，导入流和导出流，我们下面将看一下 Sqoop 作业和 Sqoop 的 API。

秘笈 9　创建一个 Sqoop 作业来处理 RDBMS

现在你已经看到导入 / 导出工具（称为 Sqoop）是如何工作的，我们准备谈论一下创建 Sqoop 作业的事情。从一个开发者甚至维护者的角度来看，一旦你正确地创建了 Sqoop 语句，你需要通过命令行来触发它。当你需要从已经改变的输入源来更新目标数据的时候，Sqoop 作业就特别有用。一般地，你需要启动一个调度作业，它要运行一晚上来提取存储在数据源中不同于前一天的新信息。一旦通过测试，计划任务调度程序就可以容易地调用命令行。

如何实现它

Sqoop 提供了命令行参数，允许我们创建一个配置好的作业。每次触发它的时候，如果没有给定命令行参数，它就会执行导入 / 导出工具。

从叙述方式转移到编码方式，让我们创建我们的第一个导入 Sqoop 作业。

1. 我们将看到第一个导入如下：

```
sqoop import-all-tables --connect jdbc:mysql://localhost/
bbdatabank --user root -P --verbose
```

2. 为创建一个作业做相同的事情，我们在我们的平台上输入下列命令：

```
sqoop job --create myimportjob  -- import-all-tables --connect
jdbc:mysql://localhost/bbdatabank --user root -P --verbose
```

它是如何工作的

我们看到命令创建了一个名为 myimportjob 的作业。该作业存储在元存储（metastore）库中，它是文件系统中的一个位置，默认情况下，它位于 $SQOOP_HOME/sqoop 目录下。

更多

一旦你创建了一个作业，可以给出 exec 参数来执行该作业，下列命令执行了 myimportjob 命令：

```
sqoop job --exec myimportjob
```

为了显示保存的作业，你可以使用下列命令：

```
sqoop job --list
```

为显示 myimport 作业的配置，使用的命令像下面这样：

```
sqoop job --show myjob
```

最后，删除保存的作业，使用下列命令：

```
sqoop job --delete myjob
```

一旦你创建了你的作业，你可以使用这种方式来管理它。万一你想在产品的环境下执行一个调度作业，你可以使用 cron 工具来调度一个作业的执行。

我们想提醒读者，除了命令行工具之外，默认情况下，作业将存储在 Sqoop 库中。当作业运行时，该库并不保存所需的密码。所以如果你要创建一个需要密码才能保存的作业，在作业执行的过程中，你将被要求输入密码。考虑一个调度作业，你应该通过修改配置文件来强制 Sqoop 存储密码。因为 Sqoop 元存储并不是最安全的一种存储，所以这也不是一种安全的行为。但是你可以通过打开 sqoop-site.xml 文件来修改这个设置；你仅仅需要取消关于下面的标签的注释：

```
<property>
  <name>sqoop.metastore.client.record.password</name>
```

```
    <value>true</value>
    <description>If true, allow saved passwords in the metastore.
    </description>
</property>
```

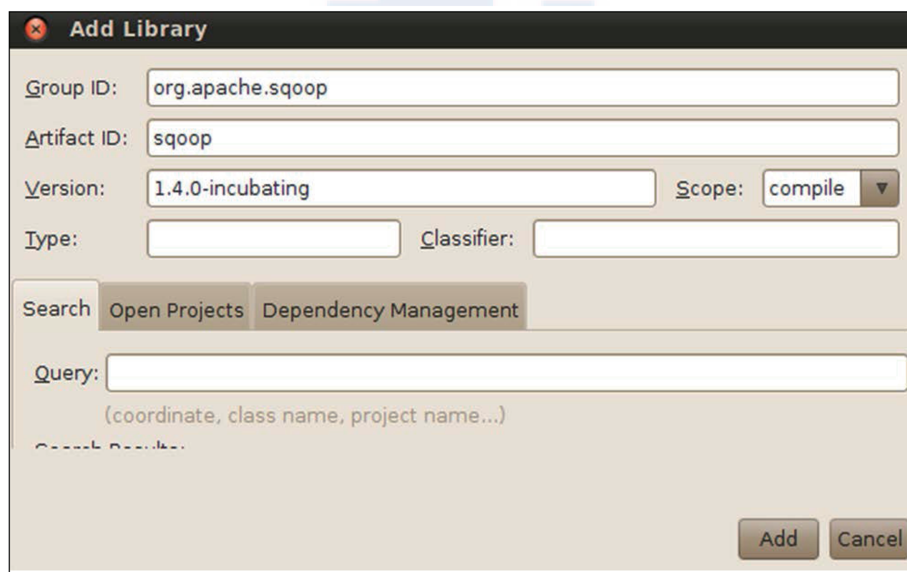
考虑到 Sqoop 主要是通过命令行工具提供某些 JAR 文件的接口，因此它能够被嵌入 Java 应用程序中。我们将在下一个秘笈中看到如何将 Sqoop 嵌入 Java 代码中。

秘笈 10　使用 Sqoop API 导入数据

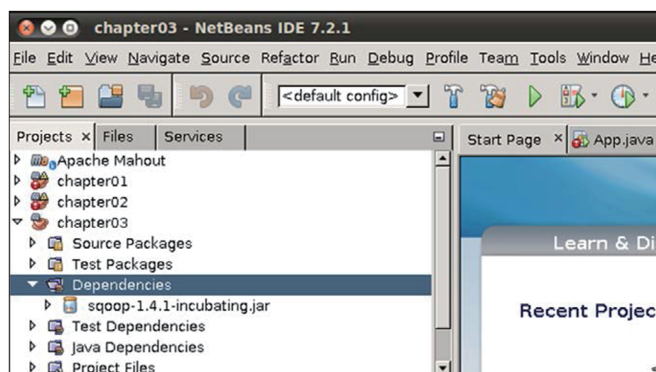
Sqoop 提供一系列 API，通过它们可以从 Java 代码中启动 Sqoop MapReduce 导入 / 导出作业。和平常一样，我们将在 NetBeans 开发环境中使用 Maven 连接正确的 JAR 包。

准备工作

和我们先前在第 1 章和第 2 章中所做的一样，打开 NetBeans 并新建一个叫 chapter 3 的 Maven 工程。然后我们使用 Maven 连接 Sqoop 包依赖，右击 Dependencies 文件夹并单击 Add Dependency 项目。在显示的对话框上，填好详细资料显示如下图所示。



然后单击 Add，你将会在你的 Dependencies 文件夹图标上看到 JAR 文件，如下图所示。



如何实现它

现在我们已经配置了我们的 Maven 工程，是时候写几行代码看看不用命令行工具如何使用 Sqoop。

我们把我们的代码加入到默认的 App.java 类中。双击它并在 main 方法中加入下列代码：

```
SqoopOptions o = new SqoopOptions();
o.setConnectString(null);
o.setExportDir("/tmp/piero");
String[] arguments = new String[10];
ImportAllTablesTool t = new ImportAllTablesTool();
int r;
r = Sqoop.runTool(arguments);
```

它是如何工作的

如你所见，我们创建了一个 SqoopOptions 对象，它对应到命令行工具中所包含的相同参数。然后，我们创建了我们所要运行的作业类型的实例。可能的作业类型如下：

- ImportAllTablesTool
- ImportTool
- ExportTool

最后一步是为每个可能的作业对象使用 runTool 方法。如果成功地完成作业，那么结果将存入一个 int 类型供编程人员解读。

我们仅提供了一个关于如何使用 Sqoop API 的简单方法。为了理解一个简单但有效的方法是如何运行的，请下载源代码看看它如何和命令行交互。

如果你打开了一个文本编辑器，你应该会看到最后一行的调用 exec 命令，如下所示：

```
exec ${HADOOP_HOME}/bin/hadoop com.cloudera.sqoop.Sqoop "$@"
```

因此，每个 Sqoop 命令的入口点都在 com.cloudera.sqoop 包的主类 Sqoop 中。我们建议读者读读本书的代码，以对场景背后发生的事情有一个更深的理解。